Szanowni czytelnicy – uczestnicy kursu procesory sygnałowe.

W trakcie rozmów po wykładach przewijały się pytania o zakres wiadomości niezbędny do spokojnego zaliczenia kursu procesorów sygnałowych. W trakcie prowadzenia jeszcze poprzedniej edycji kursu, w której przykładem architektury i możliwości były procesory rodziny TMS320C54xx opracowany został przez jednego z uczestników zbiór pytań i zagadnień, które występowały w treści różnych zaliczeń, kolokwiów zaliczeniowych i egzaminów. W kolejnych edycjach uzupełniałem i rozbudowywałem ten materiał zachowując zasadniczy układ treści autora początkowego opracowania. W obecnej jego edycji dokonałem zmian uwzględniających oparcie kursu o nowocześniejsze procesory rodziny TMS320C55xx. Przekazuję je Państwu z nadzieją, że się przyda.

Korzystając z okazji chciałbym również zwrócić uwagę na kilka elementów przygotowania i odpowiedzieć na otrzymane pytania.
Proszę zwrócić uwagę na fakt, że materiał kursu obejmuje następujące podstawowe obszary z objaśnieniem ich znaczenia i powiązań wzajemnych;

- Budowa i własności toru DSP
- Budowa procesora
    - Architektura – główne bloki procesora charakteryzujące działanie i ich powiązanie (schemat blokowy procesora)
    - Przeznaczenie głównych bloków procesora i ich ograniczenia
    - Logiczne przestrzenie zasobów procesora
    - Magistrale, odmiany i ich rola
    - Rodzaje pamięci w procesorze i ich rola
    - Mapa pamięci i jej funkcja w procesorze
    - Przestrzeń portów procesora – przeznaczenie.
    - Główne rejestry procesora i ich rola. (akumulatory, rejestry adresujące, statutowe, pomocnicze itd.)
- Działanie i mechanizmy
    - Sekwencyjność działania procesora
    - Podstawowe mechanizmy działania
        - inicjowanie – BOOT
        - porządkowanie – RESET
        - spowalnianie – READY
        - zwalnianie zasobów – HOLD
        - wtrącanie – INTERRUPT
        - bezpośredniego transferu – DMA
    - Warunki możliwe do sprawdzenia w procesorze (flagi, komparatory) i ich wykorzystanie w rozkazach warunkowych
    - Kolejka, przetwarzanie nakładkowe, zalety i wady
    - Stosy, działanie przeznaczenie, obsługa
    - Bufor kołowy działanie przeznaczenie, obsługa
    - Mechanizm buforów Ping-Pong
    - Sprzętowa realizacja pętli w procesorze
- Obliczenia, rozumienie głównych rozkazów
    - Cykle w procesorze (zegarowy, procesora, rozkazowy)
    - Główne grupy rozkazów, przykłady
    - Tryby adresacji w rozkazach, przykłady
    - Shifty, rotacje, skalowanie danych

  - Mechanizmy adresacji wspierające ukierunkowane operacje, przeznaczenie i działanie
    - Postmodyfikacja, przeznaczenie
    - BRA, (Wsteczna propagacja Carry)
    - Adresacja cyrkulacyjna
  - Usprawnianie operacji i wyników obliczeń
    - Zaokrąglanie (RND)
    - Rozszerzenie znakowe (SXMD)
    - Nasycanie (Saturation) i jego wykorzystanie
  - Reprezentacja danych w obliczeniach
    - Kodowanie dziesiętne, binarne, HEX
    - Reprezentacja InQm U2, liczb dodatnich i ujemnych
    - Stałoprzecinkowa całkowitoliczbowa (I16F0)
    - Stałoprzecinkowa ułamkowa I1Q15, I3Q13
    - Zmiennoprzecinkowa
    - Zakresy i rozdzielczość reprezentacji
    - Konwersje reprezentacji
- Podstawowe rozkazy specjalizowane – operacje w DSP i ich użycie w rozkazach
  - Mnóż i akumuluj (MAC)
  - Realizacja szybkiej transformaty Fouriera (FFT)
  - Filtr o symetrycznych współczynnikach
  - Obliczenia wartości wielomianu punkcie
- Narzędzia i środowisko developerskie
  - Elementy składowe programu asemblera, rodzaje i przeznaczenie (Elementy linii programu, sekcje, dyrektywy, zbiór konfiguracyjny linkera)
  - Podstawowy zestaw narzędzi i ich przeznaczenie (Edytor, Asembler. Kompilator, Linker, Loader, Monitor, Debugger, Emulator)
  - Sprzętowe urządzenia deweloperskie – Emulatory ICE, moduły DSK, EVM, Reference Design – cechy i przeznaczenie
  - Biblioteki wspomagające prace R&D (Research and Development)
  - Przeznaczenie i możliwości, CSL, BSL, DSPLib, IQ-math,

Często zwracano się do mnie pytając o rodzaj pytań. Podstawowymi będą pytania wymagające opisania np. cech, przeznaczenia czy budowy procesora lub jego fragmentu, sposobu działania, mechanizmu, przyczyn jakiegoś rozwiązania, charakterystyki co zyskujemy a co tracimy przez ..., podania warunków realizacji takiego czy innego rozwiązania, działania podstawowych rozkazów, składu i przeznaczenia elementów środowiska developerskiego – narzędzi wspomagających pracę projektanta/programisty, reprezentacji danych i konwersji, itp. Proszę nie spodziewać się pytań typu testowego z wyborem odpowiedzi „jak towaru z półki supermarketu". To raczej sposób sprawdzania skojarzeń i „szczęśliwej ręki" niż rzeczywistej wiedzy i rozumienia materiału.

Proszę nie pokładać przesadnego zaufania w tzw. „tabelkach", choć w materiale poświęcono im sporo miejsca. One pozwalają na sprawdzenie i przećwiczenie znajomości rozkazów, elementów reprezentacji i kodowania danych, ale są źródłem ogromnej liczby pomyłek, szczególnie w warunkach pośpiechu i stresu, zużywając mnóstwo czasu. Warto raczej walczyć z oddzielnymi pytaniami „na temat", zrozumieniem działania i budowy niż tradycyjnymi tabelami.
Zatem nie przeceniać tabelek!

Szczegóły organizacji kolokwium i dokładny podział na grupy oraz terminy zamieszczone zostaną na stronie kursu w przyszłym roku ☺

Życzę efektywnego wykorzystania dostępnych materiałów i dobrego przygotowania.

Krzysztof Kardach

OPRACOWANIE ZAGADNIEŃ DO ZALICZENIA DSP

W oparciu o pomysł opracowania Pawła Wańtowskiego
opracował dr Krzysztof Kardach 2013-01-09 00:24 (v. 4.0)

1 „Tabelki”

1.1 Tabelki „reprezentacji i konwersji arytmetycznych”

w.		DEC	HEX	BIN	12-bitowy akumulator A
	1	2	3	4	5
1	WA		0x	B	_ _ _ _ _ _ _ _ _ _ _ _
2	WB		0x	B	_ _ _ _ _ _ _ _ _ _ _ _
3	WC=WA+WB		0x	B	_ _ _ _ _ _ _ _ _ _ _ _
4	WC=WA-WB		0x	B	_ _ _ _ _ _ _ _ _ _ _ _
5	WC=WA*WB		0x	B	_ _ _ _ _ _ _ _ _ _ _ _

1. Dla liczb **A=0,75** i **B=-0,125** uzupełnić tabelę w kolumnach 2,3,4 zakładając notację **U2** i format **I1Q5**.
2. Zakładając prace procesora na słowie **6-bitowym** i kodowanie **U2**, **I1Q5**, oraz akumulator **12-bitowy** i włączony **SXMD** uzupełnij tabelę w kolumnie 5 wykonując odpowiednie rozkazy:
dla wiersza 1 **MOV #WA,<<#2,A**
dla wiersza 2 **MOV #WB,A**
Wyniki należy zapisać w notacji binarnej.
3. Wpisz do tabelki zadania 1 w kolumnie 5 (wiersze 3,4 i 5) binarną zawartość akumulatora po wykonaniu odpowiednio operacji zapisanych w kolumnie 1.
4. Jak zmieni się wynik operacji w wierszu 5, jeśli w naszym ćwiczebnym procesorze będzie ustawiony odpowiednik bitu **FRCT** (Fractional).

ad.1. UWAGA! Wyniki operacji należy zamieścić zgodnie z wymaganymi formatem i notacją.
Kodujemy **A** i **B** w kodzie **U2** (jeśli ktoś do tej pory nie wie, jak się to robi, to niech zajrzy do materiałów wystawionch na stronie kursu do materiału [Reprezentacja liczb, obliczenia i nie tylko...]. Inne przydatne materiały odnośnie systemów liczbowych i ich kodowania można znaleźć na Internecie lub w podręcznikach podstaw techniki komputerowej.

Procesor pracuje na słowie **6-bitowym**, zaś format **I1Q5** oznacza, że z tych 6 bitów jeden zostanie przeznaczony na zakodowanie części całkowitej liczby, zaś pozostałe 5 na zakodowanie części ułamkowej. (To I1Q5 to tylko przykład a nie żaden standard!. Równie dobrze może być I2Q6 czy np. I4Q12)
Tak więc pozostając dla przykładów przy I1Q5:

$$\mathbf{WA = 0{,}75_d = 0\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2} + 0\cdot 2^{-3} + 0\cdot 2^{-4} + 0\cdot 2^{-5} = 01\ 1000_b}$$
$$\mathbf{WB = -0{,}125_d}$$

Aby uzyskać **WB**, kodujemy **0,125** jak wyżej (a), następnie negujemy wszystkie bity (b) i dodajemy **1** do pozycji najmłodszego bitu (c):

(a) $\mathbf{0{,}125_d = 0\cdot 2^0 + 0\cdot 2^{-1} + 0\cdot 2^{-2} + 1\cdot 2^{-3} + 0\cdot 2^{-4} + 0\cdot 2^{-5} = 000100_b}$
(b) $\mathbf{\sim 0{,}125_d = 1\ \ldots\ 1111\ 1011_b}$
(c)

$$\begin{array}{r} 1\ldots111\ 1011 \\ +\qquad\qquad 1 \\ \hline =\mathbf{1\ldots1111\ 1100} \end{array}$$

UWAGA! Trzeba pamiętać o „1-kach z przodu") dla utrzymania wartości i znaku liczby i wyników wszelkich operacji !!! (SXM)

Zatem **WB = -0,125$_d$ =** 1 ... **11 1100$_b$**. i w 6-cio bitowym rejestrze zmieszczą się tylko wytłuszczone bity.
Teraz można wykonać następujące działania:

- **dodawanie**

```
        01 1000
+ 1 ... 1111 1100     (to liczba ujemna!)
=0 ... 0001 0100
```

Jak widać wynik jest dodatni. Zatem „0-ra" z przodu (tu zaznaczone na żółto) dla dodatnich, a „1-ki" dla ujemnych wykraczające poza rejestr trzeba pominąć, bo nie mieszczą się w rejestrze wyniku, czyli prawidłowo jest:

WA + WB = 01 0100$_b$ = 0,625$_d$

Analogicznie jest dla odejmowania (uwzględniamy tylko 6 najmłodszych bitów wyniku) i mnożenia (uwzględniamy tym razem 12 najmłodszych bitów wyniku).

- **odejmowanie** - jeżeli ktoś nie lubi bawić się w pożyczki i przenoszenie, alternatywnie zamiast wykonywać odejmowanie **A-B** można wykonać dodawanie **A+(~B)+1**, otrzymany wynik będzie taki sam (pamiętać o SXMD!):

```
   01 1000
+  00 0011
+        1
  01 1100
```

WA - WB = 01 1100$_b$ = 0,875$_d$

UWAGA: jeżeli w wyniku odejmowania dziesiętnego (dla kodowania I1Q5 U2) otrzymana zostanie liczba mniejsza od **-1**, to wykraczamy poza zakres reprezentacji! Dzieje się tak z powodu przekroczenia zakresu reprezentowanych wartości dla formatu **I1Q5 U2** (dopuszcza on minimalną wartość równą **-1**). **To jest celowy haczyk i należy napisać o tym w tabelce lub pod nią.**
Np. dla **-1,125$_d$** i reprezentacji jak wyżej na 6-ciu bitach możemy posłużyć się kalkulatorem z konwersją DEC <-> HEX. Zatem,
-1,125$_d$ * 2^5 = -36$_d$ = F ... FDC$_h$ = 1 ... 1101 1100$_b$
Zatem z zamieszczoną uwagą o wykroczeniu poza zakres reprezentacji prawidłowa odpowiedź na pytanie o zawartość rejestru (takiego 6-cio bitowego!) po wykonaniu powyższej operacji w rejestrze zmieści się **01 1100$_b$** oraz **DC$_h$** . Jeśli zaś pytanie pada o wynik kodowania **I1Q5 U2** wówczas odpowiedź jest krótka – nie da się zakodować takiej liczby I1Q5.

- **mnożenie** – by zmieścić wynik mnożenia dwóch liczb 6-cio bitowych **I1Q5 U2** potrzebny jest rejestr dwukrotnie większy. Dla uniknięcia błędów należy przygotować do operacji binarnej oba czynniki stosownie je rozszerzając znakowo co najmniej do 12 bitów (podwójnej długości) poprzez dostawienie 6 starszych bitów wypełnionych wartością taką, jaką ma najstarszy bit wagowy ze znakiem danej liczby w **U2**:

WA = 011000$_b$ → 000000 011000$_b$
WB = 111100$_b$ → 111111 111100$_b$
Po dokonaniu tego można pomnożyć **WA * WB** (tu oczywiście wygodniej będzie wymnożyć **WB * WA**):

```
                1...11 1111 1111 1100
*                    0000 0001 1000
                     0000 0000 0000
                   0 0000 0000 000
                  00 0000 0000 00
  1...1.....1 1111 1111 1111 0
  1...1....11 1111 1111 1100
            0 0000 0000 000
 …   …   …   …   …   …
```

1111...1.. 1111 **1111 1010 0000** **WA * WB = 11 1101$_b$ = -0,09375**

Wynikiem mnożenia, reprezentowanym zgodnie z przyjętą tu notacją na 6-ciu bitach, który należy zamieścić w tabelce w wierszu 5 kolumnie 4 są bity na pozycjach zaznaczonych powyżej na czerwono. (proszę zwrócić uwagę na wynikającą z położenia przecinka dodatkową 1-kę „z przodu" i fakt wyboru tylko starszych 6-ciu bitów wyniku)

Natomiast wynikiem, jaki trafi do 12-to bitowego akumulatora bez korekcyjnego przesunięcia, będzie podkreślony fragment wyniku mnożenia.

Wynik ten można również otrzymać inaczej;
-0,125 * 0,75 = -0,09375
-0,09375 * 2^5 = -3$_d$ = F … FFD$_h$ = 1 … 1111 1101$_b$

Mając już wyniki dziesiętnie i binarnie, możemy łatwo przekodować BIN na HEX. Są to dwie różne drogi rozwiązywania, obie prawidłowe i muszą dać takie same wartości liczb!.

Pierwsza: ogranicza się do 6-ciu bitów i tłumaczy dokładnie co widać;
WA = 011000$_b$ = 01 1000$_b$

1 8$_h$

czyli po prostu przekształcamy to co widać. Młodszą „czwórkę" bitów na liczbę HEX z zakresu 0-F i starszą dwójkę bitów też na liczbę HEX tyle że z zakresu 0-3:

01 1000$_2$ = 18$_h$

I tutaj nie ma problemów dla liczb z zakresu reprezentacji, dodatnich i ujemnych.

Dla przykładowej wartości; **-0,09375$_d$ = 11 1101$_b$ = 3D$_h$**sumowe wartości wag -> -1 + 0,5 + 0,25 + 0,125 + 0,03125 = -0,09375

Ale dla liczb przekraczających zakres reprezentacji potrzebna jest stosowna uwaga o której napisano wcześniej i zapis zależny od sformułowania pytania.

Zatem dla; **-1,125$_d$ = 1101 1100$_b$**
w 6-cio bitowym rejestrze będzie miejsce tylko dla **1C$_h$** no i niezbędna jest **adnotacja o przekroczeniu zakresu reprezentacji**
Zaś wartość będzie **-4 + 2** +0,5 + 0,25 + 0,125 = -1,125
(to żółte to dzięki adnotacji i tym dwóm „żółtym" bitom).

Druga: nie pomija starszych bitów a po prostu je uwzględnia. Dla dodatnich zatem uzupełnia zerami (przykład dla **WA**):

WA = 01 1000$_b$ = 0001 1000$_b$

1 8$_h$

czyli po prostu przekształcamy każdą „czwórkę" binarną na liczbę w HEX:

0001 1000$_b$ = 18$_h$

Dla ujemnych uzupełniamy zgodnie z zasadami rozszerzenia znakowego 1-kami.

Dla przykładowej wartości; **-0,09375$_d$ = 1111 1101$_b$ = FD$_h$**
stąd wartość **-4 + 2** + 1 + 0,5 + 0,25 + 0,125 + 0,03125 = -0,09375

Dla liczb przekraczających zakres reprezentacji musimy uwzględnić uzupełnienie zgodnie z wynikiem konwersji.

Zatem dla; **-1,125$_d$ = 1101 1100$_b$ = DC$_h$**
i tutaj wartość **-4 + 2** +0,5 + 0,25 + 0,125 = -1,125
(to zielone to z uzupełnienia tymi dwoma „zielonymi" bitami).

Trzeba pamiętać, że przy przekroczeniu reprezentacji pojawia się ta subtelność – jak sformułowane jest pytanie. Jeśli domaga się ono np. podania zawartości rejestru 6-cio bitowego po operacji na operandach U2 I1Q5 wówczas w odpowiedzi trzeba zawrzeć to co zmieści się w tym rejestrze (czarne bity w ostatnim przykładzie, ale koniecznie z adnotacją, że to tylko kawałek liczby z wyniku). Jeśli zaś pytanie będzie wymagało podania wyniku zakodowanego U2 I1Q5, a wynik przekracza zakres reprezentacji no to nie ma co podać. Pole powinno zostać puste ale opatrzone adnotacją, że wynik poza zakresem reprezentacji.

I tutaj jedna ważna uwaga. Przy operacjach arytmetycznych nie należy oglądać się na ustawiony lub nie bit SXMD. TUTAJ ROZSZERZENIE ZNAKOWE **MUSI CAŁY CZAS DZIAŁAĆ** BO INACZEJ NIE ZACHOWALI BYŚMY WARTOŚCI !!!

ad.2. Bit SXMD wpływa na to, jak uzupełniane są starsze bity w akumulatorze gdy ładujemy do niego liczby „krótsze" - reprezentowane na mniejszej liczbie bitów niż długość akumulatora:

- jeżeli bit SXMD jest **włączony**, to uzupełniamy w zależności od znaku, tj. gdy liczba jest dodatnia uzupełniamy zerami, a gdy ujemna - jedynkami
- gdy bit SXMD jest **wyłączony**, to zawsze uzupełniamy zerami

Mamy wykonać następujące rozkazy:
MOV #WA,<<#2,A ← załaduj **WA** do akumulatora i przesuń o **2** w lewo (* 2^{SHIFT})
MOV #WB,A ← załaduj **WB** do akumulatora

Dla **WA** będzie to wyglądać następująco (poglądowo, bo w rzeczywistości rzecz jasna nie byłoby „pustych miejsc"):

- uzupełniamy starsze bity w zależności od znaku (czyli tutaj zerami),
- ładujemy uzupełnione **WA** do akumulatora: **_ 0000 0001 1000**
- przesuwamy o 2 pozycje w lewo: **_ _ _ _ 0000 0110 00_ _**
  a na najmłodszych **zawsze** wstawiamy zera: **0000 0110 0000**

Postępując analogicznie (oczywiście tym razem bez przesuwania), dla **WB** mamy:

1111 1111 1100

Można jeszcze rozpatrzyć podobny przykład np. z przesunięciem o 4 ;
MOV #WB,<<4,A ← załaduj **WB** do akumulatora i przesuń o **4** w lewo (* 2^{SHIFT})
- uzupełniamy starsze bity w zależności od znaku (czyli tutaj zerami),
- ładujemy uzupełnione **WA** do akumulatora: **_ 1111 1111 1100**
- teraz przesuwamy o 4 pozycje w lewo, wychodzące poza rejestr najstarsze bity obcinamy a najmłodsze bity uzupełniamy zerami: **_ 1111 1100 0000**

Warto tutaj zrobić jeszcze jedną uwagę. Najpierw rozszerzamy liczbę a potem przesuwamy. I jeszcze jedna uwaga, dla stałych wprowadzanych do akumulatorów dysponujemy tutaj tylko przesunięciem SHFT w lewo. Przesunięcie w lewo i w prawo SHIFTW [-32 / +31] dysponujemy jedynie przy przesłaniach z pamięci i rejestrów.

ad.3. Dla dodawania i odejmowania postępujemy podobnie, jak w punkcie poprzednim, pamiętając o włączonym bicie SXMD. Inaczej mówiąc ładujemy wyniki operacji do akumulatora i odpowiednio uzupełniamy zerami lub jedynkami w zależności od znaku. W przypadku mnożenia wynik jest dłuższy i po prostu przepisujemy do akumulatora najmłodsze 12 bitów otrzymane w wyniku pisemnego mnożenia binarnego, czyli w naszym wypadku tę podkreśloną część wyniku ze strony 3:

1111 1010 0000

ad.4. Bit FRCT powoduje skasowanie „nadmiarowego" znaku wyniku operacji mnożenia liczb, co dzieje się poprzez przesunięcie liczby o **1** w lewo, czyli jeśli wynik mnożenia będzie w rejestrze produktu jak niżej:

1111 1010 0000

to po przesunięciu o jeden w lewo uzyskamy w akumulatorze:

1111 0100 0000

Powstałą wskutek przesunięcia „pustą" pozycję na najmłodszym bicie **zawsze** uzupełniamy zerem.
Jeszcze uwaga dla mających kłopot z rozumieniem określenia „nadmiarowego znaku". Proponuję policzyć pozycje po przecinku obu czynników, następnie określić położenie przecinka w wyniku (dokładnie tak, jak to się robi przy mnożeniu pisemnym liczb dziesiętnych) i natychmiast ujawni się dodatkowe miejsce przed przecinkiem i wynikła z tego konieczność korekcyjnego przesunięcia w lewo realizowanego za sprawą bitu FRCT. Bit FRCT nie zmienia innych wyników operacji ani przesłań do akumulatorów!

I to już wszystko, po prawidłowym uzupełnieniu tabelka „arytmetyczna" powinna zawierać następujące wartości:

A = 0,75, B = -0,125

w.		DEC	HEX	BIN	**ACC** 12-bitowy akumulator
	1	2	3	4	5
1	WA	0,75	0x18	01 1000 B	0000 0110 0000 B
2	WB	-0,125	0x3C	11 1100 B	1111 1111 1100 B
3	WC=WA+WB	0,625	0x14	01 0100 B	0000 0001 0100 B
4	WC=WA-WB	0,875	0x1C	01 1100 B	0000 0001 1100 B
5	WC=WA*WB	-0,09375	0x3D	11 1101 B	1111 1010 0000 B
A po uwzględnieniu FRCT akumulator będzie zawierał;					1111 0100 0000 B

Poniżej 4 inne tabelki z innych grup z hipotetycznymi rozkazami do wykonania na stałych w kolumnie 5

dla wiersza 1 **MOV #WA<<3,ACC**
dla wiersza 2 **MOV #WB,ACC**
dla wiersza 3 **ADD #WA,#WB,ACC**
dla wiersza 4 **SUB #WA,#WB,ACC**
dla wiersza 5 **MPY #WA,#WB,ACC**

A = -0,875, B = 0,5

w.		DEC	HEX	BIN	**ACC** 12-bitowy akumulator
	1	2	3	4	5
1	WA	-0,875	0x24	10 0100 B	1111 0010 0000 B
2	WB	0,5	0x10	01 0000 B	0000 0001 0000 B
3	WC=WA+WB	-0,375	0x34	11 0100 B	1111 1111 0100 B
4	WC=WA-WB	-1,375*	0xD4; 0x14*	01 0100* B	1111 1101 0100 B*
5	WC=WA*WB	-0,4375	0x32	11 0010 B	1110 0100 0000 B

*przekroczenie zakresu reprezentacji poniżej granicznej -1;.
UWAGA, licząc binarnie trzeba pamiętać o rozszerzeniu znakowym (uzupełnieniu „jedynek" z przodu) inaczej otrzymamy BŁĘDNY WYNIK ~~0,625~~
*widać tylko zawartość 6-ciu najmłodszych bitów bo starsze „obcięło" (nie weszły)
*prosze zauważyć, że to co nie mieściło się w rejestrze (przekroczenie zakresu) będzie jużreprezentowane w dwa razy większym rejestrze!!!

A = 0,125, B = -0,75

w.		DEC	HEX	BIN	**ACC** 12-bitowy akumulator
	1	2	3	4	5
1	WA	0,125	0x04	000100B	0000 0010 0000 B
2	WB	-0,75	0x28	101000B	1111 1110 1000 B
3	WC=WA+WB	-0,625	0x2C	101100B	1111 1110 1100 B
4	WC=WA-WB	0,875	0x1C	011100B	0000 0001 1100 B
5	WC=WA*WB	-0,09375	0x3D	111101B	1111 1010 0000 B

A = -0,25, B = 0,375

w.		DEC	HEX	BIN	**ACC** 12-bitowy akumulator
	1	2	3	4	5
1	WA	-0,25	0x38	11 1000 B	1111 1100 0000 B
2	WB	0,375	0x0C	00 1100 B	0000 0000 1100 B
3	WC=WA+WB	0,125	0x04	00 0100 B	0000 0000 0100 B
4	WC=WA-WB	-0,625	0x2C	10 1100 B	1111 1110 1100 B
5	WC=WA*WB	-0,09375	0x3D	11 1101 B	1111 1010 0000 B

A = 0,75, B = -0,25

w.		DEC	HEX	BIN	**ACC** 12-bitowy akumulator
	1	2	3	4	5
1	WA	0,75	0x18	01 1000 B	0000 1100 0000 B
2	WB	-0,25	0x38	11 1000 B	1111 1111 1000 B
3	WC=WA+WB	0,5	0x10	01 0000 B	0000 0001 0000 B
4	WC=WA-WB	1*			0000 0010 0000 B
5	WC=WA*WB	-0,1875	0x3A	11 1010 B	1111 0100 0000 B

*przekroczenie zakresu reprezentacji; Tej liczby nie da się przedstawić na 6-ciu bitach.
ALE w akumulatorze o podwójnej długości da się przedstawić bo ten 6-ty bit nie będzie już skrajnym bitem z informacją o znaku jak jest to w „krótkim rejestrze"!

1.2 Tabelka „wprawki rozkazowe"

Sprowadza się do odpowiedniej interpretacji rozkazów – (pod tabelką przykładowa instrukcja interpretacji krok po kroku). Górna tabelka poza zadaniem wartości bitów ustalających sposób interpretacji rozkazów zawiera dane o zawartości

fragmentów pamięci danych. Zwracam uwagę na wskaźnik strony DP, który może być podany a czasami trzeba go określić w oparciu o adres pamięci w tabelce. **Wszystkie liczby wstawione do tabelki podczas rozwiązywania są w HEX, a nie w DEC!**

Dane:
CPL=0
CMPT=0
Adr./Dane HEX

DP=0	
60	60
61	40
62	
Adres	**Wartość**

DP=2	
200	120
201	70
202	20
Adres	**Wartość**

DP=6	
300	130
301	80
302	100
Adres	**Wartość**

	Program	**AC1**	**AC2**	**DP**	**AR0**	**AR1**	**AR2**
1	MOV #0,DP			**0**			
2	MOV #2,AR0				**2**		
3	MOV #200h,AR1					**200**	
4	MOV #300h,AR2						**300**
5	MOV @0x61,AC1	**40**					
6	ADD *AR1+,A	**160**				**201**	
7	SUB @60h,AC1,AC2		**100**				
8	ADD *AR1+,AC2,AC1	**170**				**202**	
9	MOV #6,DP			**6**			
10	ADD @2,AC1	**270**					
11	ADD *AR2+,AC1	**3A0**					**301**
12	SUB *AR2+,AC1	**320**					**302**
13	SUB #64,AC1	**2E0**					
14	ADD *AR2-0,AC1,AC2		**3E0**				**300**
15	SUB *AR2,AC2,AC1	**2B0**					
16	MOV #160,AR0				**A0**		
17	ADD *AR1-0,AC1,AC1	**2D0**				**162**	
18	MOV AC1,*AR1-					**161**	

Przykładowa interpretacja kolejnych wierszy tabeli:

1. Załaduj liczbę binarną 0 do DP. (włączamy w ten sposób 0-wą stronę pamięci danych)
2. Załaduj liczbę binarną 2 do AR0.
3. Załaduj liczbę 200h (albo inaczej 0x200) do AR1.
4. Załaduj liczbę 0x300 do AR2.
5. Załaduj wartość komórki o adresie 0x61 do A (patrz nad tabelką).
6. Dodaj do akumulatora A zawartość komórki pamięci danych o adresie równym zawartości AR1 (zawartość AR1 to 0x200, a odpowiadająca temu adresowi zawartość komórki to 0x120 – patrz nad tabelką), dodaj do tego zawartość A, zapisz wynik w A i potem inkrementuj zawartość AR1.
7. Odejmij od akumulatora A zawartość komórki pamięci danych o adresie 0x60 (adres 0x60 zawartość równa 0x60 – patrz nad tabelką) i wynik zapisz w B.
8. Dodaj do akumulatora B zawartość komórki pamięci danych o adresie równym wartości AR1 (zawartość AR1 to 0x201, a odpowiadająca temu adresowi wartość komórki to 0x70 – patrz nad tabelką), a zapisz wynik w A, potem inkrementuj zawartość AR1.
9. Załaduj liczbę 6 do DP. (włączamy w ten sposób 6-tą stronę pamięci danych)
10. Mając DP=6 (zrobiliśmy to w wierszu 9) patrzymy do tabelek nad główną tabelą i szukamy DP=6 (pierwsza mała, górna tabelka od prawej strony). @2 w wierszu 10 mówi, że interesuje nas adres 2 komórki na 6-tej stronie pamięci danych (6 * 0x80 + 2 = 0x302), a wiec w tym przypadku adres 302h. Zatem zawartość komórki o adresie 0x302 czyli 0x100 dodajemy do A i wynik zapisujemy w A.

11. Zawartość komórki pamięci danych o adresie równym zawartości AR2 (zawartość AR2 to 0x300, a odpowiadająca temu adresowi zawartość komórki danych to 0x130 – patrz nad tabelką) dodaj to do zawartości akumulatora A i zapisz wynik w A, potem inkrementuj zawartość AR2.
12. Zawartość komórki pamięci danych o adresie równym zawartości AR2 (zawartość AR2 to 0x301, a odpowiadająca temu adresowi wartość komórki to 0x80 – patrz nad tabelką), odejmij od zawartości akumulatora A i zapisz wynik w A, następnie inkrementuj zawartość AR2.
13. Podaną w rozkazie (adresacja natychmiastowa) wartość dziesiętną 64 (co daje 40h) odejmij to od zawartości akumulatora A i zapisz wynik w A.
14. Zawartość komórki pamięci danych o adresie równym zawartości AR2 (zawartość AR2 to 0x302, a odpowiadająca temu adresowi zawartość komórki to 0x100 – patrz nad tabelką). Dodaj to do zawartości akumulatora A i wynik zapisz w B. Teraz *AR2-0 oznacza: od tego, co jest w AR2, odejmij to, co jest w AR0 (0x302-2=0x300) i zapisz w AR2.
15. Zawartość komórki pamięci danych o adresie równym zawartości AR2 (zawartość AR2 to 0x300, a odpowiadająca temu adresowi zawartość komórki pamięci danych to 0x130 – patrz nad tabelką). Odejmij to od B i zapisz wynik w A.
16. Załaduj liczbę dziesiętną 160 (co daje A0h) do AR0.
17. Zawartość komórki pamięci danych o adresie równym zawartości AR1 (zawartość AR1 to 0x202, a odpowiadająca temu adresowi zawartość komórki to 0x20 – patrz nad tabelką). Dodaj to do A i zapisz wynik w A. Teraz AR1-0 oznacza: od tego, co jest w AR1, odejmij to, co jest w AR0 i zapisz w AR1 (202h-A0h=162h).
18. To, co jest w A, załaduj do komórki o adresie równym zawartości AR1.

UWAGA: w powyższej tabelce znajduje się kolejny drobiazg. W obszarze danych (nad tabelką) jest puste miejsce pod adresem 0x62 sugerujące miejsce czekające na wstawienie czegoś. Tymczasem z 18 wiersza tabelki wynika, że trzeba załadować wartość znajdującą się w akumulatorze AL. do komórki o adresie 0x162 (i nie jest to omyłka). W takim przypadku można obok tabelki narysować poglądowo fragment pamięci z komórką o adresie 0x162, do której ma trafić wartość zawarta w akumulatorze, np. tak jak obok:

.	
.	
160	
161	
162	2D0
.	
.	

Poniżej wypełnione tabelki dla innych dwóch grup z 2004 roku.

Dane:

	DP=0		DP=2		DP=4	
	Adres	**Wartość**	**Adres**	**Wartość**	**Adres**	**Wartość**
CPL=0	60	60	100	120	200	130
CMPT=0	61	40	101	60	201	50
Adr./Dane HEX	62	**240**	102	20	202	100

	Program	**AC1**	**AC2**	**DP**	**AR0**	**AR1**	**AR2**
1	MOV #0,DP			**0**			
2	MOV #2,AR0				**2**		
3	MOV #100h,AR1					**100**	
4	MOV #200h,AR2						**200**
5	MOV @0x61,AC1	**40**					
6	ADD *AR1+,A	**160**				**101**	
7	SUB @60h,AC1,AC2		**100**				
8	ADD *AR1+,AC2,AC1	**160**				**102**	
9	MOV #4,DP			**4**			
10	ADD @1,AC1	**1B0**					
11	ADD *AR2+,AC1	**2E0**					**201**
12	SUB *AR2+,AC1	**290**					**202**
13	SUB #64,AC1	**250**					
14	ADD *AR2-0,AC1,AC2		**350**				**200**
15	SUB *AR2,AC2,AC1	**220**					
16	MOV #160,AR0				**A0**		
17	ADD *AR1-0,AC1,AC1	**240**				**62**	
18	MOV AC1,*AR1-					**61**	

Dane:

	DP=0		DP=2		DP=4	
	Adres	**Wartość**	**Adres**	**Wartość**	**Adres**	**Wartość**
CPL=0	60	60	100	120	200	130
CMPT=0	61	140	101	30	201	70
Adr./Dane HEX	62	**310**	102	20	202	100

	Program	**AC1**	**AC2**	**DP**	**AR0**	**AR1**	**AR2**
1	MOV #0,DP			**0**			
2	MOV #2,AR0				**2**		
3	MOV #100h,AR1					**100**	
4	MOV #200h,AR2						**200**
5	MOV @0x61,AC1	**140**					
6	ADD *AR1+,A	**260**				**101**	
7	SUB @60h,AC1,AC2		**200**				
8	ADD *AR1+,AC2,AC1	**230**				**102**	
9	MOV #4,DP			**4**			
10	ADD @1,AC1	**2A0**					
11	ADD *AR2+,AC1	**3D0**					**201**
12	SUB *AR2+,AC1	**360**					**202**
13	SUB #64,AC1	**320**					
14	ADD *AR2-0,AC1,AC2		**420**				**200**
15	SUB *AR2,AC2,AC1	**2F0**					
16	MOV #160,AR0				**A0**		
17	ADD *AR1-0,AC1,AC1	**310**				**62**	
18	MOV AC1,*AR1-					**61**	

Dane:

	DP=0		DP=2		DP=4	
	Adres	**Wartość**	**Adres**	**Wartość**	**Adres**	**Wartość**
CPL=0	60	60	100	120	200	130
CMPT=0	61	140	101	30	201	470
Adr./Dane HEX	62	**FF10**	102	20	202	100

	Program	**AC1**	**AC2**	**DP**	**AR0**	**AR1**	**AR2**
1	MOV #0,DP			**0**			
2	MOV #2,AR0				**2**		
3	MOV #100h,AR1					**100**	
4	MOV #200h,AR2						**200**
5	MOV @0x61,AC1	**140**					
6	ADD *AR1+,A	**260**				**101**	
7	SUB @60h,AC1,AC2		**200**				
8	ADD *AR1+,AC2,AC1	**230**				**102**	
9	MOV #4,DP			**4**			
10	ADD @112,AC1	**2A0**					
11	ADD *AR2+,AC1	**3D0**					**201**
12	SUB *AR2+,AC1	**FF FFFF FF60***					**202**
13	SUB #64,AC1	**FF FFFF FF20**					
14	ADD *AR2-0,AC1,AC2		**20**				**200**
15	SUB *AR2,AC2,AC1	**FF FFFF FEF0**					
16	MOV #160,AR0				**A0**		
17	ADD *AR1-0,AC1,AC1	**FF FFFF FF10**				**62**	
18	MOV AC1,*AR1-					**61**	

*Uwaga, od tego momentu wkraczamy w liczby ujemne

I jeszcze jedna uwaga. Wyniki obliczeń podlegają wpływowi ustawienia bitu OVM. W zamieszczonych przykładach nie było powodu tym się zajmować bo nie zbliżaliśmy się do granic nasycania. Ale problem istnieje!

2 Pytania z testów - opracowanie

2.1 Wymień główne cechy wyróżniające procesory sygnałowe od innych procesorów i mikrokontrolerów.

- sprzętowa jednostka mnożąca (MAC)
- szybki shifter (Barrel Shifter) do skalowania danych
- sprzętowe nasycanie i zaokrąglanie
- sprzętowy mechanizm realizacji pętli poprzez repetycję rozkazów i bloków rozkazów
- specjalizowane rozkazy do przetwarzania sygnałów (FIRS, SUBC, POLY,…)
- jednostki arytmetyczne dla obliczeń na adresach
- liczne, specjalizowane rejestry do adresacji pośredniej
- rozbudowany mechanizm modyfikacji adresów wspomagający specyficzne korekty adresów np. dla potrzeb FFT
- sprzętowy mechanizm obsługi buforów kołowych
- sprzętowe, wewnątrz struktury procesora wsparcie mechanizmu debugowania i emulacji
- zwielokrotnienie i specjalizacja magistral w tym osobne magistrale danych i programu procesora
- rozbudowane systemy pamięci notatnikowych (cache)
- znaczne moce obliczeniowe wyrażane w MIPS i FLOPS w zależności od typu i specjalizacji procesora

- wydzielone magistrale specjalizowane do obsługi strumienia danych sygnału (zwykle magistrale szeregowe)
- małe obudowy (BGA, TQFP) przy małym poborze mocy

2.2 Jaka jest w procesorach C55xx rola rejestrów statusowych i dlaczego jest ich aż cztery?

Rejestry te służą do zachowania informacji o stanie pracy procesora i wybranych ustawieniach. Jest ich aż cztery (ST0_55, ST1_55, ST2_55, ST3_55) gdyż zachodzi potrzeba przechowania znacznej liczby danych, których nie da się przechować w mniejszej liczbie rejestrów.

2.3 Jakie zmiany w architekturze wprowadzone w kolejnych generacjach procesorów pozwoliły na zwiększenie szybkości wykonania programu?

Na przykładzie różnic między C54xx, C55xx, C6000:
- zwielokrotnienie zasobów
    - MAC i ALU
    - Akumulatorów
    - dodatkowe generatory adresów
    - dodatkowe jednostki przetwarzania równoległego (do 8-miu)
- poszerzenie magistral
- poszerzenie listy rozkazów / procedur specjalizowanych
- rozbudowa mechanizmów dostępu do danych i programu
- rozbudowa mechanizmu cache wielopoziomowego
- zwiększenie równoległości przetwarzania
- wydłużenie słowa adresowego (architektura WLIV)
- wprowadzenie sprzętowych jednostek zmiennoprzecinkowych operacji
- wprowadzenie specjalizowanych jednostek – koprocesorów - np. do obsługi różnych standardów interfejsów, procedur (np. FFT) czy peryferii (np. sensory CCD, eQEP).
- Zwielokrotnienie rdzeni procesorów DSP

2.4 Od czego można uzależnić przebieg programu w procesorach rodziny C55xx?

Generalnie sekwencyjny przebieg rozkazu modyfikują skoki. W tym skoki warunkowe mają szczególne znaczenie, bo realizowane są w zależności od spełnienia lub nie warunku lub warunków. Pytanie dotyczy wskazania od czego można uzależnić przebieg programu czyli jakie warunki jesteśmy w stanie wykorzystać w tych warunkowych skokach, a zatem;
Wiele stanów jest wykrywane i sygnalizowane flagami;
- ACOVx czyli przekroczenia w każdym z akumulatorów
- C – Carry – przeniesienie w użytym akumulatorze,
- TC – bit, gdzie trafiają wyniki operacji logicznych

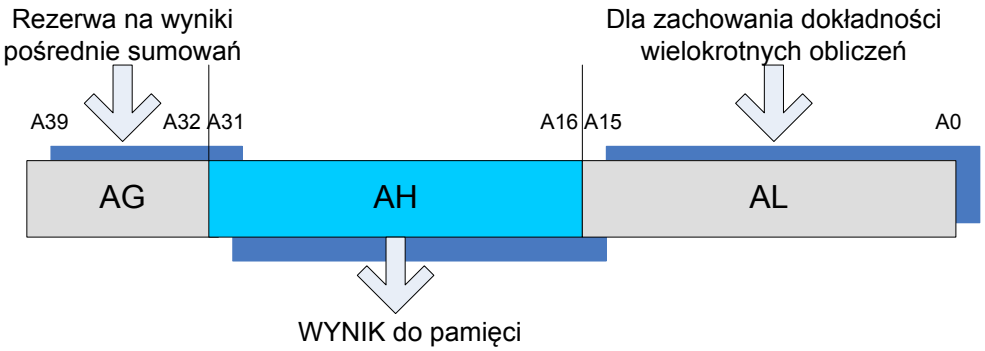
Inne elementy mogą być wykrywane bezpośrednio (zwykle komparatorami);
- Zawartość akumulatorów i jej relacja względem zera,
- Zawartość rejestrów tymczasowych Tx i ich relacja względem zera,
- Zawartość rejestrów adresowych ARx i ich relacja względem zera,
- Stan testowanego dowolnego bitu w pamięci danych (trafi do TC)
- Zawartość całej komórki w pamięci danych

Warunki te mogą być wykorzystywane w takich rozkazach sterujących przebiegiem programu jak:

- instrukcji skoku (BCC)
- instrukcji wywołania procedur (CALLCC)
- instrukcji powrotu (RETCC)
- instrukcji odpowiedzialnych za repetycję (RPTCC)

2.5 Dlaczego w procesorach sygnałowych rejestr akumulatora jest ponad dwa razy większy od rozmiaru słowa, jakim pracują?

Aby przyjąć wynik mnożenia liczb binarnych, potrzebny jest akumulator będący dwukrotnie większy niż rozmiar słowa. Wynik mnożenia odbierany do przechowania w pamięci danych znajduje się w takiej sytuacji na starszej części akumulatora – AH. Młodsza część akumulatora – AL młodsze 16 bitów – ma za zadanie zapewnienie większej rozdzielczości dla sumowania wyników mnożenia i uniknięcia kumulowania błędów na skutek przedwczesnego ograniczania rozdzielczości reprezentacji. Ta młodsza część akumulatora może uzyskać swój wpływ na wartość wyniku poprzez operację zaokrąglania (Rounding). Z kolei dodawanie może doprowadzić do wyniku wykraczającego ponad 32 bity akumulatora stąd rezerwę do sumowania lub wyników pośrednich sumowania zapewniają bity AG - GUARD (jest ich osiem).



2.6 Co to jest przetwarzanie nakładkowe, na czym polega i czemu służy?

Przetwarzanie nakładkowe, albo kolejka (z angielskiego pipelining), jest to sposób wykorzystania zasobów procesora do realizacji rozkazów tak, by żaden jego fragment „nie stał bezczynnie”. Uwarunkowane jest podziałem realizacji rozkazu na kolejne fazy wykonywane w pojedynczych cyklach procesora i możliwościami bloków przetwarzających procesora oraz magistral transportu danych i rozkazów. Polega on na równoczesnym wykonywaniu różnych faz kolejnych rozkazów programu. Przykładowo, wykonując fazę Pre-Fetch dla jednej instrukcji, procesor może jednocześnie wykonywać fazę Fetch poprzedniej instrukcji, fazę Decode dla jeszcze wcześniejszej instrukcji itd. Dzięki temu rozkazy pobierane do kolejki są wykonywane quazi równolegle – po jednej fazie z każdego z kolejnych rozkazów – jak gdyby cały jeden rozkaz w jednej fazie. Liczba poziomów kolejki, jej głębokość, zależy od liczby faz na które podzielone zostało wykonywanie rozkazu. W procesorach rodziny ‘C54xx rozkaz jest podzielony na 6 fa realizacji, stąd kolejka ma 6 poziomów działania. W ten sposób przy wypełnionej kolejce w jednym cyklu procesora wykonywanych jest równocześnie 6 różnych faz – czyli „jeden cały rozkaz”. Fakt, że każda z faz pochodzi z innego rozkazu nie zmienia faktu, że w rozliczeniu czasu wykonywania programu na jeden cykl procesora przypada wykonanie kompletnego rozkazu dając przyspieszenie realizacji programu.

Trzeba mieć jednak świadomość, że bez kolejki na wykonanie pojedynczego rozkazu o rozmiarze jednego słowa – na cały cykl rozkazowy wykonywany bez kolejki - potrzeba 6-ciu cykli procesora, po jednym cyklu na każdą z faz.
Zatem technika ta nie skraca czasu wykonywania pojedynczego rozkazu ale dzięki nakładaniu na siebie rozkazów pozwala na skrócenie wykonywania sekwencji rozkazów i przyspieszenie wykonania całego programu.
W przetwarzaniu kolejkowym pobierając zawartość kolejnych komórek z pamięci programu natrafia się na trudności związane z wykonywaniem skoków. Typowy skok zakłóca ciągłość kolejki i wymaga jej ponownego napełnienia rozkazami po skoku. Innym problemem jest „konfliktu kolejki", gdy kolejny rozkaz chce wykorzystać wynik działania bezpośrednio poprzedzającego go rozkazu, a poprzednik jeszcze nie ukończył działania. Metodą zaradzenia tym kłopotom jest reorganizacja programu lub wprowadzenie dodatkowego opóźnienia (np. rozkazem NOP).

2.7 Dlaczego pojedyncza magistrala zewnętrzna procesora DSP stanowi istotne ograniczenie dla jego szybkości działania?

Jeżeli pamięć programu i danych będą umiejscowione na zewnątrz procesora, to pojedyncza magistrala zewnętrzna może transportować tylko jeden obiekt. Albo kod rozkazu, albo jedną daną. To zaś uniemożliwia wykorzystanie walorów kolejki i szybkość realizacji programu spada. Może to spowodować obniżenie efektywności, o co najmniej 50%.

2.8 Dlaczego w procesorze DSP stosuje się wiele równoległych magistral transportowych?

Dlatego, że inaczej nie można wykorzystać walorów przetwarzania nakładkowego (kolejki). Jest ono tylko wtedy efektywne, gdy możliwe jest pobieranie w tym samym czasie zarówno operandów do przetwarzania (nawet dwóch równocześnie) jak też i kodu/kodów kolejnych rozkazów oraz odsyłanie wyniku operacji do pamięci. Wielość magistral do równoległych transferów, praca z pamięcią podwójnego dostępu wraz ze specjalnymi technikami wykonywania rozkazów (rozkazy z opóźnieniem) i modyfikacja kolejności rozkazów lub dwufazowa kolejka jak w c55xx przekłada się na szybszą realizację całego programu.

2.9 Co to jest DARAM i dlaczego jest korzystna w procesorach DSP C55xx?

DARAM (Double Access RAM) jest to pamięć zezwalająca na 2 dostępy w jednym cyklu procesora w każdym z bloków pamięci. Oznacza to, że różne jednostki obliczeniowe MAC, czy CPU jak i wewnętrzne zespoły mogą dokonywać odczytu i zapisu w tym samym cyklu. Część rozkazów może być efektywnie wykonywana tylko, gdy ich operandy rozmieszczone są w DARAM (np. rozkazy z grupy MAC, rozkazy równoległe). Istotne jest ponadto, że pamięć DARAM z przestrzeni pamięci danych można przełączyć (uwidocznić) do przestrzeni pamięci programu (to w starszych procesorach, poprzez ustawienie bitu OOVLY) zaś w nowych przygotowana ciągła pamięć może być dostępna „z dwóch stron", zarówno jako pamięć danych jak i programu.

2.10 Wymień tryby adresacji stosowane w rodzinie procesorów TMS320C55xx i podaj przykłady rozkazów stosujących je.

Adresacja	**Przykład**	**Przeznaczenie, zalety**
Natychmiastowa (Immediate)	MOV #10,AC1	- operand bezpośrednio w kodzie rozkazu - użyteczne do inicjalizacji [*umieść 10d w akumulatorze A*] `{K => A}`
Absolutna (Absolute)	MOV AC1,*(y)	- używa pełnego 23-bitowego adresu dowolnej komórki [*zachowaj młodszą część akumulatora A pod adresem y w pamięci danych*] `{(A(L)) _ Smem(y)}`
Pośrednia (Indirect)	MOV *AR1,AC1	- adresem operandu jest zawartość aktywnego rejestru (ARi) użyta jako wskaźnik [*zapisz w akumulatorze A wartość z komórki pamięci danych spod adresu zawartego w rejestrze AR1*] `{(Smem(AR1)) => A}`
Bezpośrednia (Direct)	MOV @x, AC1	- adresacja względem wskaźnika strony -DP albo wskaźnika stosu –SP (decyduje bit CPL) [*zapisz w akumulatorze A zawartość komórki z pamięci danych spod adresu otrzymanego ze złożenia x z DP lub sumy x z SP*] `{(Smem(x⊥DP)/(x+SP)) => dst}`

Kołowa (Circular)	ADD AR0+,AC1	- gdy używana adresacja cyrkulacyjna modyfikacja AR0 będzie w opszarze zadeklarowanego rejestru kołowego
		[dodawaj kolejne wartości do akumulatora AC1]
		`{AC1 + (AR0) => AC1`
		`(AR0 + 1 => AR0)`
		`Stosownie z ST2_55, BK03, BSA01}`

2.11 Jak rozumiesz i co określa pojęcie trybu adresacji?

Tryb adresacji jest sposobem definiowania dostępu do operandu w treści rozkazu (podawania adresu w rozkazie). Określa on, w jaki sposób instrukcje sięgają do swoich operandów w pamięci. Wyróżniamy następujące tryby adresacji:

- natychmiastowy np. MOV #10,AC1
- absolutny MOV AC1, *(y)
- pośredni MOV *AR1,AC1
- bezpośredni MOV @x,AC1
- Cyrkulacyjny ze specyficznym użyciem rejestrów adresujących

Tryby adresacji i ich rozumienie i sprawność wykorzystania to kluczowy element efektywnego programowania przetwarzania we wszystkich procesorach nie tylko sygnałowych.

2.12 Wymień podstawowe sposoby modyfikacji zawartości rejestrów adresowych procesorów C55xx i podaj ich przykładowe przeznaczenie.

Opcja	**Składnia**	**Sposób realizacji**
Bez modyfikacji	*ARn	ARn bez zmian

Post-Inkrement / Post-Dekrement	*ARn+ *ARn-	post inkrementacja o 1 post dekrementacja o 1
Post-Indeksowana	*(ARn+AR0) *(ARn–AR0)	post inkrementacja o zawartość AR0 post dekrementacja o zawartość AR0 podobnie z rejestrem T0 i T1
Post-Mod-Kołowa (circular)	*(ARn+AR0%) *(ARn-AR0%)	kołowo post inkrementacja o zawartość AR0 kołowo post dekrementacja o zawartość AR0
Post-z odwr. Bitów (Bit-Reversed or Reverse Carry Propagation)	*(ARn+AR0B) *(ARn-AR0B)	post inkrementacja o AR0 z odwróc. Bitów (albo wsteczną propagacją Carry) post dekrementacja o AR0 z odwróc. Bitów (albo wsteczną propagacją Carry)
Pre-modyfikacja	*ARn(AR0)	chwilowe pre *(ARn+AR0), bez zminy ARn!
Podobnie z CDP		Ale nie względem AR0 a stałych (#K16)

Przykładowe zastosowania:

- inkrement/dekrement – dostęp do tablic, wektorów, sygnałów
- kołowe – dostęp do tablic i wektorów ale ze sprzętową kontrolą przemieszczania się w buforze (zapewnia automatyczny skok na/przez początek/koniec bufora), obsługa buforów współczynników i próbek dla filtrów, transformat i transferu danych
- z odwróceniem bitów - dla szybkiej transformaty Fouriera (FFT) i innych transformat wykorzystujących własności symetrii funkcji sin/cos

2.13 **Co to są sekcje programu i do czego są używane?**

Sekcje to fragmenty programu zawierające jednorodne obiekty; kod, stałe, zmienne lub układy we/wy. Są one zdefiniowane za pomocą dyrektyw w zbiorach źródłowych.
Sekcje dzielimy wg. zawartości na;

- sekcja inicjalizowana (kod programu, predefiniowane stałe),
- sekcja nieinicjalizowana (rezerwacja obszarów pamięci na zmienne czy stałe)

i wg. opisu na

- sekcja nazwana (opatrzone nazwą)
- sekcja nienazwana (bez nazwy)

Sekcje są umieszczane przez linker we wskazanych obszarach pamięci zgodnie z zapisem zbioru konfiguracyjnego. Sekcje o tych samych nazwach łączone są we wspólne obszary ułatwiając organizację danych w pamięci.

2.14 **Co to jest dyrektywa asemblera i do czego służy?**

Dyrektywa asemblera jest to polecenie definiujące mu sposób traktowania danego fragmentu programu. Są elementem sterowania asemblacją programu. Nie są tłumaczone na rozkazy programu a jedynie uruchamiają sposób działania asemblera. Dyrektywy mogą służyć np. do zdefiniowania sekcji w zbiorach źródłowych, uaktywnienia własności asemblera, itd. Są one poleceniami tekstowymi i zaczynają się od kropki.
Przykładami dyrektyw mogą być:

```
.mmregs          ; włącza predefiniowane nazwy rejestrów MMR
.sect „kot”      ; kończy poprzednio zdefiniowaną sekcję i otwiera nową
```

```
                                ; inicjalizowaną i nazwaną „kot” sekcję na kod programu
                                ; lub dane
        .text                   ; kończy poprzednio zdefiniowaną sekcję i otwiera nową
                                ; inicjalizowaną i nazwaną sekcję na kod programu
        .bss    test,n          ; kończy poprzednio zdefiniowaną sekcję i otwiera nową
                                ; nieinicjalizowaną sekcję o nazwie test na n słów danych
        .usect „pies”, n        ; kończy poprzednio zdefiniowaną sekcję i otwiera nową
                                ; nieinicjalizowaną i nazwaną „pies” sekcję dla danych
                                ; rezerwując dla nich n słów w pamięci danych
tab     .word  4, 0x13, 66h     ; kończy poprzednio zdefiniowaną sekcję i otwiera nową
                                ; inicjalizowaną sekcję dla trzech wartości 4, 13h, 66h
                                ; zaczynającą się od adresu „tab”
```

2.15 Objaśnij zadania linkera w środowisku programów do generacji kodu procesora DSP.

Linker łączy plik *.obj i generuje docelowy plik wyjściowy *out. Rozmieszcza on i łączy jednoimienne sekcje w obszarach pamięci wskazanych w zbiorze/poleceniach konfiguracyjnych. Linker może generować różne, pomocne w analizie i uruchamianiu programu zbiory np. *.map – mapę pamięci, *.lst – pełnego listingu programu, *.hex – zbiór dla programatora pamięci, itd. Zajmuje się on rozmieszczeniem relokowalnych zbiorów *.obj a w nich symboli i sekcji, by przypisać je do ostatecznych adresów oraz decyduje o zewnętrznych powiązaniach między plikami wejściowymi i bibliotekami. Do prawidłowego działania linkera niezbędny jest zbiór konfiguracyjny linkera - Linker Command File (w CCS ma on rozszerzenie .cmd).

2.16 Wymień czynniki decydujące o szybkości realizacji programu w DSP.

a) wynikające z budowy procesora

- częstotliwość taktowania procesora
- przetwarzanie nakładkowe
- zwielokrotnienie magistral
- rozkazy specjalizowane i ukierunkowane na aplikacje
- zastosowanie adresacji kołowej lub z odwracaniem bitów
- rozkazy skoków z opóźnieniem
- łączone warunki dla skoków i operacji warunkowych
- wykonywanie rozkazów w trybie repetycji
- zaawansowana obsługa pośrednich wyników operacji
- operacje dwusłowowe
- wielkość pamięci wewnętrznej, szczególnie DARAM
- ilość i sposób wykorzystania przerwań oraz ich ewentualne kolizje z trybami repetycji

b) wynikające ze sposobu przygotowania programu

- wykorzystanie wymienionych wyżej możliwości sprzętowych
- podział programu pomiędzy asembler i języki wysokiego poziomu
- rozmieszczenie danych w pamięciach SARAM / DARAM, pamięci zewnętrznej i/lub wewnętrznej

2.17 Omów sposoby realizacji pętli i stosowane tam rozkazy.

Do realizacji pętli mogą zostać wykorzystane następujące rozwiązania:

- repetycja pojedynczego rozkazu realizowana instrukcją RPT, pozwala na powtórzenie instrukcji od 1 do 65536 razy. RPT można wykonać równolegle z zerowaniem akumulatora dla „czystego" początku sumowania (odpowiednik RPTZ w rodzinie C54xx).
  RPT n → n+1 powtórzeń
- repetycja bloku rozkazów realizowana za pomocą instrukcji RPTB. Pozwala ona na powtórzenie bloku instrukcji od 1 do 65536 razy. Liczbę obiegów pętli ustala się przed wywołaniem repetycji ustalając zawartość BRC (Block Repeat Counter). BRC=n → n+1 powtórzeń

  Fakt repetycji bloku rozkazów z rozróżnieniem dwóch poziomów zagłębienia jest sygnalizowany odpowiednimi stanami bitów w rejestrze CFCT, dla uniknięcia niszczenia zawartości rejestrów określających warunki repetycji (odpowiednik flagi BRAF dla uproszczonego mechanizmu repetycji w procesorach C54xx).
- instrukcje skoku warunkowego, wykonujące skok tylko wtedy, gdy spełniony jest dany warunek (w przeciwnym razie wykonanie programu przechodzi do następnej instrukcji). Wyróżniamy dwie instrukcje skoku warunkowego:
  - BC: przeładowuje PC bezpośrednim adresem, gdy spełniony jest warunek. Zazwyczaj wykorzystywane do testów arytmetycznych wykonywanych na zawartości akumulatora lub testowania flag.
  - Warunkiem takim może być również zawartość wybranego rejestru ARx, który może służyć równocześnie jako licznik obiegów pętli. (odpowiedniość rozkazu BANZ w rodzinie C54xx).
- instrukcje skoku bezwarunkowego – dla pętli bez końca.

2.18 Co to są tryby repetycji i czemu służą w procesorach DSP rodziny C'55xx?

Tryb repetycji polega na powtarzaniu rozkazu lub bloku rozkazów. W trybie tym dzięki sprzętowej obsłudze licznika pętli nie tracimy czasu na rozkazy sprawdzające licznik i realizujące skok. Stąd pętle takie są bardziej efektywne, tylko użyteczna część pętli zajmuje czas wykonania.

Repetycja pojedynczego rozkazu:

- uruchamiana instrukcją RPT pozwala na powtórzenie następnej instrukcji od 1 do 65536 razy.

RPT n → n+1 powtórzeń. Niestety pętli takiej nie można przerwać przerwaniem!

Repetycja bloku rozkazów:

- uruchamiana za pomocą instrukcji RPTB „etykieta"

Pozwala na powtórzenie od 1 do 65536 razy bloku instrukcji od inicjującego rozkazu do rozkazu opatrzonego etykietą. Liczba przebiegów zadana jest zawartością BRC (Block Repeat Counter) plus jeden; BRC=n → n+1 powtórzeń. Zakres pętli zadawany jest zawartościami rejestrów RSA – adres początku pętli, REA – adres końca pętli.

2.19 W jaki sposób i po co programista może określać/zmieniać położenie tablicy wektorów przerwań (początków procedur przerwań)?

Po resecie sprzętowym procesor nadając wartość rejestrom IVPD i IVPH równą 0xFFFF będzie sięgał do tablicy wektorów przerwań zaczynającej się od adresu 0xFFFF00. Domyślnie, dla takiej sytuacji tablica wektorów przerwań jest lokowana w zakresie adresów od FFFF00h do FFFFFFh w przestrzeni pamięci programu.

Można przygotować inną/inne tablice w przestrzeni pamięci programu, zaczynające się od adresów równych (IVPD)*256 / (IVPH)*256 i wskazać ją procesorowi do użycia poprzez nadanie odpowiedniej zawartości rejestrów a następnie wykonanie **programowego** reset (czyli rozkazu RESET).
Mechanizm taki jest zaimplementowany z tego powodu, by użytkownik mógł reorganizować strukturę przerwań swego programu w zależności od potrzeb.
Np. gdy nie chce używać domyślnych wektorów przerwań rezydujących w pamięci ROM układu może przesunąć wskazanie tablicy wektorów do dowolnej 256-słowowej przestrzeni w pamięci programu i tam przygotować jej własną wersję.

2.20 Co to jest i czemu służy w procesorach rodziny C'55xx IVPD?

IVPD (czyli DSP Interrupt Vector Pointer) to 16-to bitowy rejestr procesora umieszczony pod adresem 0x000049. Jego zawartość stanowi najstarsze 16 bitów adresu w tablicy wektorów przerwań. Uzupełniona kodowanym na 5 bitach numerem przerwania i najmłodszymi trzema bitami 000 tworzy adres początkowy w tablicy wektorów przerwań procesora, czyli adres położenia pierwszego wektora w tej tablicy. Jest to wektor przerwania RESET złożony z pierwszych ośmiu bajtów programu jego obsługi – startu procesora. Po sprzętowym RESET procesora IVPD = 0xFFFF co lokuje początek tej tablicy pod adresem 0xFFFF00. Programista może przełączyć procesor do odczytywania tablicy wektorów przerwań z innego miejsca w pamięci programu przez zmianę zawartości IVPD i wykonanie programowego RESET. Rejestr IVPD wskazuje tablicę dla wektorów przetwań 0 – 15 i 24 - 31

2.21 Co to jest i czemu służy w procesorach rodziny C'55xx IVPH?

IVPH (czyli Host Interrupt Vector Pointer) to 16-to bitowy rejestr procesora umieszczony pod adresem 0x00004A. Jego zawartość stanowi najstarsze 16 bitów adresu w tablicy wektorów przerwań. Uzupełniona kodowanym na 5 bitach numerem przerwania i najmłodszymi trzema bitami 000 tworzy adres początkowy w tablicy wektorów przerwań procesora, czyli adres położenia pierwszego wektora w tej tablicy. Jest to wektor przerwania RESET złożony z pierwszych ośmiu bajtów programu jego obsługi – startu procesora. Po sprzętowym RESET procesora IVPH = 0xFFFF co lokuje początek tej tablicy pod adresem 0xFFFF00. Programista może przełączyć procesor do odczytywania tablicy wektorów przerwań z innego miejsca w pamięci programu przez zmianę zawartości IVPH i wykonanie programowego RESET. Rejestr IVPH wskazuje tablicę dla wektorów przerwań 16 – 23

2.22 Dla procesora 'C5515 podaj, co to jest, co może zawierać, gdzie znajduje się i do czego służy tablica wektorów przerwań?

Tablica wektorów przerwań jest to obszar w pamięci programu procesora, gdzie umieszczone są ośmiobajtowe wektory przerwań, będące początkami procedur obsługi przerwań odpowiadających danym lokacjom w tablicy. Domyślnie rozpoczyna się ona pod adresem 0xFFFF00 ale programista może wskazać procesorowi inne jej położenie w pamięci programu, odpowiednio przygotowując wcześniej tam jej zawartość i modyfikując zawartość rejestrów IVPD i IVPH przed programowym RESET. Rejestry te zawierają 16 najstarszych bitów adresu położenia początku tablicy wektorów przerwań (8 młodszych bitów musi być zerami). W ten sposób adresy tych możliwych tablic wyrażają się (pma)=IVPD*256d (dla adresacji bajtowej!)
Tablica ta służy wiązaniu odpowiednich przerwań procesora z obsługującymi je procedurami obsługi – czyli procedurami reakcji na fakt wystąpienia danego przerwania albo rozkazu INTR/TRAP.
[V3.x CPU- str. 2.24],

2.23 Co to jest przerwanie?

Przerwanie (ang. interrupt) – to mechanizm służący synchronizacji przebiegu programu z niezależnymi od programu zdarzeniami. Służą do tego sygnały przerwań informujące o wystąpieniu zdarzenia, procedury reagowania na zdarzenia – obsługi tych zdarzeń, oraz mechanizmy maskowania i szeregowania ważności tych zdarzeń – ich priorytetów. Decydują one, czy zgłoszenie zdarzenia zostanie zauważone (obsłużone) przez procesor a w przypadku równoczesnego zgłoszenia kilku zdarzeń rozstrzygają, które z nich należy obsłużyć najpierw.
Zdarzenia mogą być wewnętrzne np. zmiany w zasobach wewnętrznych procesora (przepełnienie licznika, koniec transmisji danych, koniec przetwarzania wewnętrznego przetwornika A/C, itp.) albo zdarzenie zewnętrzne, które generują sygnały doprowadzone do wejść przerwań zewnętrznych (INT0, INT1, ... INTn). Żądanie przerwania wyrażone sygnałem może wystąpić w dowolnym momencie (w dowolnej fazie cyklu procesora) niezależnie od programu. Wymaga ono zarejestrowania. Przed przystąpieniem do oceny ważności oczekujących przerwań i obsługi najważniejszego z nich wymagane jest dokończenia właśnie realizowanego rozkazu. Pojawienie się niezamaskowanego przerwania powoduje wstrzymanie aktualnie wykonywanego programu i wykonanie przez procesor kodu procedury obsługi przerwania (ISR). Po zakończeniu obsługi – wykonania procedury ISR procesor wraca do wykonywania przerwanego programu.
Mówiąc o przerwaniach należy starannie formułować wypowiedzi, bo w technicznym żargonie często terminem „przerwania" określa się zarówno sygnały jak i same programy obsługi przypisane zdarzeniom czy też same zdarzenia obsługiwane tym mechanizmem.

2.24 Co to jest procedura obsługi przerwania i jakie są jej główne cechy?

Procedura obsługi przerwania (ISR) to przygotowany fragment programu zawierający sekwencję rozkazów opisujących sposób reagowania procesora (systemu) na występujące zdarzenie – jego sygnał.
Dla prawidłowego działania procedury obsługi przerwania należy poza jej przygotowaniem ustawić globalną maskę przerwań - INTM, odpowiedni bit indywidualnej maski danego przerwania w rejestrze masek przerwań - IMR, wartość wskaź-

nika stosu - SP i umieścić początek procedury obsługi przerwania w tablicy wektorów przerwań – przygotować odpowiedni wektor.
Procedura obsługi przerwania jest zatem programem wywoływanym po ustawieniu flagi danego przerwania, który po dokończeniu wykonywanego rozkazu i przy dopuszczeniu przerwania odpowiednimi stanami masek globalnej i indywidualnej zostanie uruchomiony począwszy od własnego wektora w tablicy wektorów przerwań. Po dostrzeżeniu sygnału przerwania (zawsze badanego na końcu każdego rozkazu), stwierdzeniu dopuszczalności przerwania (odpowiedniego ustawienia masek) a przed jej przywołaniem procesor:

- Automatycznie kasuje flagę przerwania,
- automatycznie odsyła na stos jedynie zawartość rejestru PC (czyli adres następnego rozkazu do wykonania po ukończeniu obsługi przerwania),
- automatycznie ustawia globalną maskę przerwań INTM co daje zablokowanie przerwań. (Zatem inne przerwania nie mogą wystąpić bez zgody programisty),
- tworzy przypisany dla zidentyfikowanego przerwania adres początkowy wektora w tablicy wektorów przerwań (pierwszego rozkazu procedury ISR) i umieszcza go w PC,
- odczytuje pierwszy rozkaz spod utworzonego adresu (4-ro słowowy wektor w tablicy zawiera zwykle skok do dalszego ciągu programu procedury ISR),
- dla przerwań zewnętrznych sygnalizuje rozpoczęcie wykonywania procedury linią INTA procesora ,

Procedurę ISR charakteryzuje zwykle

- na początku procedury konieczność zachowania (zwykle na stosie) stanu rejestrów procesora używanych w trakcie jej działania i odtworzenie ich zawartości na końcu procedury. (Zachowywanie rejestrów na stosie i pobieranie ich na końcu odbywają się w odwrotnej kolejności.)
- kończenie procedury rozkazem RET lub RETI by odblokować system przerwań.

2.25 Co wiąże, a co różni indywidualną maskę przerwania i flagę przerwania?

I flaga i maska są występującymi w różnych rejestrach bitami (przerzutnikami). Wiąże je to samo przerwanie, tyle że maska jest bitem blokującym lub dopuszczającym obsługę przerwania a flaga jest bitem zgłoszenia żądania obsługi przerwania.
Od strony operacyjnej łączy pewne podobieństwo. Flaga jest ustawiana sprzętowo za sprawą wystąpienia zewnętrznego sygnału, choć może być również ustawiona programowo. Jednak kasowana jest WYŁĄCZNIE sprzętowo na początku obsługi przerwania i po RESET. Zaś maska może być ustawiana i kasowana programowo a dodatkowo ustawiana jest sprzętowo przy rozpoczynaniu obsługi przerwania i po RESET.

2.26 Czego dotyczą operacje „context save” i „context restore” w procedurach ISR i jakim podlegają zasadom?

Operacje te dotyczą zachowania i odtworzenia stanu kluczowych rejestrów procesora oraz tych, które będą używane w trakcie ISR. Dotyczy to rejestrów statusowych procesora ST0, ST1, PMST oraz jeśli uruchomiona możliwość innego przerwania (zagłębiania przerwań) - również masek przerwań IMR. Potrzeba zachowania stanu rejestrów używanych w procedurze obsługi przerwania wynika z konieczności powrotu do przerwanego programu głównego z takim stanem procesora jaki został zastany przez przerwanie.

Operacje te realizowane są za pomocą następujących rozkazów:

Rozkaz	**Opis**
PSH	Przesłanie na szczyt stosu danych z pamięci, rejestru lub pary rejestrów z korektą (odpowiednim zmniejszeniem) zawartości wskaźnika stosu SP adresującego w obszarze stosu (Rozkaz o bardzo różnorodnym działaniu)
POP	Pobranie ze szczytu stosu danych i skierowanie do pamięci, rejestru lub pary rejestrów z korektą (odpowiednim zwiększeniem) zawartości wskaźnika stosu SP adresującego w obszarze stosu (Rozkaz o bardzo różnorodnym działaniu)
PSHBOTH	Przesłanie na szczyt stosu danych z wybranego akumulatora lub rozszerzonego rejestru adresującego (XARn) z korektą (odpowiednim zmniejszeniem) zawartości wskaźnika stosu SP adresującego w obszarze stosu
POPBOTH	Pobranie ze szczytu stosu danych i odtworzenie zawartości wskazanego akumulatora lub rozszerzonego rejestru adresowego (XARn) z korektą (odpowiednim zwiększeniem) zawartości wskaźnika stosu SP adresującego w obszarze stosu (Rozkaz o bardzo różnorodnym działaniu)

Trzeba pamiętać o odwrotnej kolejności pobierania danych ze stosu do kolejności zapisywania na stosie.
[V3.x InstSet_RG; str. 495 i 506]

2.27 Co to jest stos i jaka jest zasada jego działania i do czego on służy?

Stos jest to fragment obszaru pamięci danych, na którym adresację realizuje rejestr wskaźnika stosu SP. Stos jest realizacją rejestru typu LIFO i charakteryzuje się odwrotną kolejnością pobierania danych ze stosu do kolejności ich zapisywania.
Stos służy głównie do zachowania i ochrony stanu procesora w trakcie realizacji procedur obsługi przerwania (context save/context restore), zachowania adresów procedur przywoływanych rozkazami CALL, przekazu parametrów do procedur i funkcji, itd.
Wskaźnik stosu - SP wskazuje zawsze ostatnią zajętą komórkę stosu (czyli ostatnią odesłaną na stos daną), zatem dla;
Dla CALL: PC → *--SP
dla odesłania stanu PC na stos najpierw musimy zmniejszyć stan rejestru SP o jeden by wskazać wolną komórkę pamięci na stosie a potem dopiero odesłać daną na wskazaną pozycję.
a dla RET: *SP++ → PC
odczytujemy (popularnie pobieramy) zawartość szczytu stosu (TOS) i kierujemy do PC a potem zwiększamy wskaźnik stosu.
Dla zdefiniowania stosu należy:

1. Zadeklarować nieinicjalizowaną sekcję odpowiedniego rozmiaru, rezerwującą wystarczający obszar pamięci dla stosu.
2. Sekcję tę skierować (umieścić) za pośrednictwem zbioru konfiguracyjnego linkera w pamięci (najlepiej w pamięci wewnętrznej)
3. Zainicjować wskaźnik stosu (SP) by wskazał "szczyt stosu +1":

2.28 Jakie warunki i gdzie można sprawdzać w procesorze C55xx, czego one dotyczą i jakie rozkazy mogą wykorzystywać ich wyniki?

Trzeba zauważyć, że możliwość sprawdzania różnych warunków pozwala na realizację rozgałęzień programu poprzez wykorzystanie skoków warunkowych. W procesorze 'C5515 te możliwości są bardzo szerokie i obejmują nie tylko nadzór zawartości akumulatorów i ich wzajemnych relacji ale również zawartości rejestrów ARx, Tx, indywidualnych bitów w pamięci oraz bitów portów. Realizacji tej kontroli służą liczne flagi, komparatory oraz własności rozkazów.
W procesorach 'C55xx można sprawdzać następujące warunki dotyczące zawartości akumulatorów:

ACx	ACx==0,	ACx!=#0,	ACx<#0,	ACx>#0,	ACx<=#0,	ACx>=#0,
ARx	ARx==0,	ARx!=#0,	ARx<#0,	ARx>#0,	ARx<=#0,	ARx>=#0,
Tx	Tx==0,	Tx!=#0,	Tx<#0,	Tx>#0,	Tx<=#0,	Tx>=#0,

flag sygnalizujących wyniki operacji:

overflov(ACx) => ACOVx=1, !overflow(ACx) => ACOVx=0,
CARRY => CARR=1, !CARRY => CARRY=0,
TCx => TCx=1, !TCx => TCx=0,
wynik 1 („prawda") dla operacji TC1 i TC2 połączonych relacjami AND (&), OR (|) i XOR (^)

Na wartości tych flag wpływają wyniki operacji/rozkazów:

- logiczne AND, OR, XOR (bitowo, również na rejestrach i pamięci danych)
- testowania pojedynczych bitów i pól (B…)
- testowania relacji młodszych i starszych części akumulatorów (CMP)
- testowania relacji między całymi akumulatorami, rejestrami, komórkami pamięci
- testowania stanu linii portów

Warunki te mogą być wykorzystywane w rozkazach warunkowych skoków (BCC, RCC, RPTCC, RETCC), odwołań (CALLCC).

2.29 Do czego służy w procesorach DSP zegar (timer)?

Zegary (timery) w DSP można zastosować do:

- generację przerwań po ustalonym programowo czasie (np. dla RTC)
- generowania impulsów zewnętrznych po ustalonym programowo czasie
- sterowania generacją impulsów PWM
- realizacji przetwornika C/A
- pomiar czasu trwania funkcji czy innych procesów software'owych
- Zliczania zdarzeń zewnętrznych lub wewnętrznych w systemie
- generację impulsów i pomiar ich szerokości
- generacji zdarzeń synchronizujących dla DMA, A/C, C/A i innymi peryferiami.

2.30 Co odróżnia standardowy port szeregowy od McBSP w C55xx?

McBSP to wielokanałowy buforowany port szeregowy. Od standardowego portu szeregowego odróżniają go następujące cechy:

- pełny, dwukierunkowy bezpośredni interfejs do układu codec i innych urządzeń szeregowych.
- podwójne buforowanie dla nadawania i potrójne dla odbioru transmisji
- zdolność realizacji wielu standardów komunikacji szeregowej
- praca wielokanałowa maksymalnie do 128 kanałów

- długość słowa: 8-, 12-, 16-, 20-, 24-, 32-bit
- wewnętrzna generacja zegara/ramek z SGR (Sample Rate Generator)
- transmisja 8-bitowa danych z możliwością wyboru pierwszeństwa LSB lub MSB

Ponadto, podobnie jak standardowy port szeregowy, McBSP zapewnia:
- maksymalna szybkość bitowa: 1/2 CPU Clock Rate
- wbudowany kompanding wg. praw μ lub A
- programowana polaryzacja zegara / ramek
- potrafi sygnalizować wszystkie typowe błędy i statusy

2.31 Na czym polega konfigurowanie do pracy peryferii w procesorach DSP?

Polega na zdefiniowaniu odpowiednich parametrów (głównie przy użyciu dokumentacji i CSL - Chip Support Library) i wypełnieniu rejestrów konfiguracyjnych (określeniu ich zawartości), które mogą nadzorować pracę programu. Na CSL składają się:

- struktury danych (myConfig, etc.) - wartości do umieszczenia w rejestrach
- funkcje (DMA_config, etc.) - pozwalają na inicjowanie i zarządzanie zasobami
- makra (DMA_OPT_RMK(), etc.) - zapewniają dostęp z wysokiego poziomu do operacji niskiego poziomu

CSL zapewnia dwie podstawowe możliwości:
- programowanie peryferii
- kierowanie zasobami (utrzymanie sposobu użycia środków)

2.32 Do czego są szczególnie przydatne w procesorach DSP kanały DMA i z czym głównie współpracują?

Kanały DMA w procesorach DSP współpracują głównie z McBSP oraz D/A. Można mówić o ich szczególnej przydatności ze względu na to, że:
- DMA dla przesłań może sięgać do każdego zasobu
- prowadzą transfer danych bez zaangażowania CPU, a zatem odciążają procesor
- posiadają autoinit (automatyczne ustawienie następnego kanału do transferu)
- transfer można synchronizować np. z 20 zdarzeniami w C55xx
- każdy z kanałów dysponuje FIFO by zapis mógł wyprzedzać odczyt (gdy zasoby docelowe są zajęte)
- przy wydzieleniu kanału z użyciem DMA obsługiwanych może być do 128 kanałów
- pozwala na niezależny wybór wielu kanałów (słów), które mają być transmitowane i odbierane
- elastycznie indeksowana adresacja DMA pozwala na sortowanie każdego kanału do oddzielnego bufora

Jednym z najlepszych zastosowań DMA w procesorach DSP jest powiązany z sortowaniem automatyczny transfer danych między portami McBSP a RAM zawartym w układzie. Upraszcza to zarządzanie i obsługę wielokanałowych portów szeregowych przez procesor. W miarę rozbudowy mechanizmu DMA w kolejnych generacjach procesorów jego funkcjonalność wzrasta dzięki rozbudowie mechanizmów synchronizacji i wiązania w łańcuchy (sekwencje) powiązanych operacji.

2.33 Co to jest i do czego służy emulator (ICE) procesora DSP?

Emulator to urządzenie, którego zadaniem jest zapewnić kontrolę nad praca procesora DSP. Umożliwia w zasadzie bardzo podobne funkcje jak debugger. Różnica polega na tym, że w przypadku debuggera funkcje te są zapewniane i obsługiwane przez program uruchomiony na docelowym procesorze, natomiast emulator częściowo działa na procesorze, wykorzystując jego zasoby m.in. do komunikacji i do nadzoru a częściowo na komputerze nadrzędnym - Host. Wykorzystanie emulatora przekłada się na ułatwienie prac uruchomieniowych oprogramowania procesora, poprawę możliwości diagnostycznych błędów programu działającego z pełną szybkością i z podłączonymi peryferiami a dzięki temu możność sprawdzenia programu w prawdziwych warunkach i szybsze przygotowanie produktu.

2.34 Dla 12-to bitowej reprezentacji liczb kodowanych U2 i I3Q9 określ zakres (MAX, MIN) i rozdzielczość reprezentacji (LSB).

Należy skorzystać z następujących wzorów:
min $-(2^{I-1})$
max $2^{I-1}-2^{-Q}$
rozdz. 2^{-Q}

Zatem:
min $-(2^2) = -4$
max $2^2-2^{-9} = 4 - 0{,}001953125 = 3{,}998046875$
rozdz. $2^{-9} = 0{,}001953125$

2.35 Po co i jak stosuje się zaokrąglenie wyniku?

W procesorach sygnałowych rodziny ‘C5000 wyniki operacji umieszczane są w akumulatorach 40 bitowych. Wynik operacji odsyłanej dalej zwykle mieści się na starszej części akumulatora (AH/BH). Najstarsza część – bity ochronne – (AG/BG) stanowią rezerwę dla sumowania wyników pośrednich operacji a część młodsza (AL/BL) ma zapewnić odpowiednią dokładność obliczeń wyników pośrednich akumulatorze.
W odbieranym wyniku z 16-to bitowej części starszej akumulatora AH można uwzględnić końcówkę wyniku zawartą w części młodszej AL właśnie poprzez użycie wbudowanego w procesor mechanizmu zaokrąglania wyniku. W praktyce oznacza to dodanie do akumulatora wartości 0x00.0000.8000, dzięki czemu jeśli zawartość części AL akumulatora przekracza ½ LSB części AH akumulatora wówczas jego zawartość zostanie zaokrąglona (AH=AH+1 albo inaczej A=A+0x8000). Mechanizm ten uruchamiamy specjalizowanym rozkazem RND albo odpowiednio modyfikowanymi rozkazami operacji arytmetycznych np. MACR, MPYR, LDR itp.
Zaokrąglanie np. w obliczeniach pętli filtrów stosuje się zazwyczaj dla wyniku ostatniej operacji.

2.36 Co w procesorach określa pojęcie rozszerzenia znakowego i dlaczego jest ono tak istotne w DSP?

Procesory sygnałowe dla zachowania odpowiedniej precyzji obliczeń zawsze dysponują akumulatorami, co najmniej dwukrotnie większymi od rozmiaru słowa,

którym pracują. W przypadku rodziny procesorów 'C5000 pracującej na słowie 16-to bitowym akumulatory mają rozmiar 40-to bitowy.
Rozszerzenie znakowe to mechanizm pozwalający procesorowi w takiej sytuacji na zachowanie znaku danej ładowanej do większego rejestru. Operacja ta realizowana jest automatycznie i może być włączana za pomocą bitu SXMD – Sign eXtention Mode – umieszczonego w rejestrze statusowym ST1_C55.
Zasada działania SXMD jest następująca:
SXMD = 1 → liczby ujemne dopełniane są na starszych bitach jedynkami, zaś dodatnie zerami.
SXMD = 0 → brak dopełnienia znakowego, starsze bity pozostają zwykle bez zmian.
Obsługa włączania rozszerzenia znakowego wygląda następująco:

```
BSET SXMD       ;sign-extension mode ON
BCLR SXMD       ;sign-extension mode OFF
```

2.37 Co to jest Saturation on Store (SST)?

Saturation on Store (SST) - jest to operacja nasycania wyniku przy zapamiętywaniu. Włączana jest i wyłączana za pośrednictwem bitu SST w rejestrze statusowym PMST (ST3_C55.0). Gdy SST=1, włączone jest nasycanie wartości z akumulatora przed odesłaniem do pamięci. Nasycanie jest wykonywane po operacji przesunięcia (jeśli rozkaz tego wymaga). Trzeba jednak podkreślić, że odsyłając do pamięci „nasyconą" zawartość nie nasycamy zawartości akumulatora!. Podczas użycia SST zatem wykonywane są następujące operacje:

- 40-bitowa wartość jest przesuwana (w prawo lub lewo) w zależności od instrukcji).
- 40-bitowa wartość jest nasycana do wartości zależnej od bitu statusowego M40. Gdy M40=0 do wartości 32-bitowej a sposób nasycenia zależy od bitu SXMD.
  Jeśli SXMD = 0, generowana jest następująca 32-bitowa wartość:
  - FFFF FFFFh, jeśli wartość jest większa niż FFFF FFFFh
  Jeśli SXMD = 1, generowana jest następująca 32-bitowa wartość:
  - 7FFF FFFFh, jeżeli wartość jest większa niż 7FFF FFFFh
  - 8000 0000h, jeżeli wartość jest mniejsza niż 8000 0000h
  Gdy M40=1 nasycanie odbywa się na poziomie wartości 40-bitowej zatem do wartości 0x7F FFFF FFFF i 0x80 0000 0000
- Otrzymana zawartość jest przesyłana do pamięci w sposób zależny od instrukcji
- Ważne jest, że wszystkie te operacje na zawartości akumulatora są tylko tymczasowe dla przygotowania wartości do zachowania w pamięci. Zawartość akumulatora po zakończeniu operacji pozostaje taka jak na początku wykonywania rozkazu (niezmieniona), podobnie jak ACOVx.

Zatem jest to coś w rodzaju ograniczenia napięcia zasilania w układach analogowych, które nie pozwala na wyjście napięciem wyższym, ponad poziom zasilania.

2.38 Czy „Saturation On Store" to jedyny sposób realizacji nasycania?

Nie. Do dyspozycji jest również rozkaz nasycania akumulatora wskazanego w rozkazie. SAT ACx. Istotną różnicą w stosunku do „nasycania przy zapamiętywaniu" jest to, że tutaj następuje trwałą zmiana zawartości akumulatora – nasycenie. W przypadku SST zmieniał (nasycała) się jedynie wartość odsyłana do pamięci a akumulator pozostawał niezmieniony dla dalszych operacji.

2.39 Co to jest Overflow Mode Saturation Mode in D Unit i co zmienia w pracy procesora jego włączenie?

Overflow Mode jest trybem nadzoru przepełnienia zakresu. Włącza się go / wyłącza poprzez modyfikację bitu sterującego nasycaniem. W rodzinie C54xx był to bit OVM w rejestrze statusowym (ST1.9). W procesorach rodziny C55xx za uruchomienie takiego sposobu pracy odpowiada co do pozycji w rejestrach statusowych ten sam bit (ST1_55.9) tym razem opatrzony oznaczenie SATD (Saturation Bit for Unit D). Z tą zmianą nazewnictwa związana jest również zmiana sposobu działania polegająca na uzależnieniu poziomu nadzoru przepełnienia i nasycania od stanu bitu M40 (na poziomie 32 lub 40 bitu akumulatorów). SATD i M40 determinują, jaka jest zawartość akumulatora gdy dojdzie do przepełnienia:
gdy SATD=0, wyniki w akumulatorze nie podlegają ograniczaniu. Wyniki obejmują wszystkie 40 bitów w akumulatorze (operacje modulo-40)

Gdy SATD=1 a M40=0, wyniki w akumulatorze są ograniczane pomiędzy wartościami maksymalnymi; dodatnią 0x00.7FFF.FFFF i ujemną 0x00.8000.0000, nie dopuszczając do przekroczenia zakresu. W związku z tym wynik obliczeń nie przekracza 32-bitów przy przekroczeniu zakresu.

Gdy SATD=1 a M40=1, wyniki w akumulatorze są ograniczane pomiędzy wartościami maksymalnymi; dodatnią 0x7F_FFFF_FFFF i ujemną 0x80_0000_0000, nie dopuszczając do przekroczenia zakresu. W związku z tym wynik obliczeń nie przekracza 40-bitów.

Warto dodać, że w procesorach C55xx występuje również niezależnie sterowana operacja ograniczenia wartości (nasycania) w ALU jednostki A. Steruje tym bit SATA i w przypadku jego ustawienia (BSET SATA) wyniki operacji w ALU jednostki A w przypadku przekroczenia w (w stronę maksymalnych dodatnich) górę ograniczone zostaną do wartości 0x7FFF, a w przypadku przekroczenia w stronę maksymalnych ujemnych na 0x8000.

2.40 Jak włącza się w procesorach rodziny C55xx DSP tryb ograniczenia wartości (Overflow Mode)?

Włączenie to można wykonać kilkoma sposobami.

- Albo poprzez ustawienie indywidualnego bitu SATD (OVM w C54xx) posługując się rozkazem „ustaw pojedynczy bit nazwa_bitu" :

  BSET SATD

  Odpowiednio, bit OVM może być kasowany (ustawiony na 0) rozkazem „skasuj pojedynczy bit nazwa_bitu":

  BRST SATD

  Należy pamiętać o odpowiednim dla wymaganego poziomu ograniczenia ustawieniu bitu M40
- Albo poprzez ustawianie całej zawartości rejestru statusowego przy użyciu rozkazów operacji logicznych z odpowiednio przygotowanymi maskami (z zerem lub jedynką na pozycji naszego kasowanego / ustawianego bitu;

  OR #0x0200, ST3_55 ; SATD=1

lub jego skasowania
AND #0x0FDFF, ST3_55 ; SATD=0

2.41 Czym się różni przepełnienie od nasycenia?

Przy aktywnym nasycaniu podczas przekroczenia zakresu w akumulatorze ustawiana jest maksymalna lub minimalna możliwa wartość, natomiast po przepełnieniu, na skutek posługiwania się rejestrem o skończonej długości (operacja modulo ...) następuje „przekręcenie" się zawartości, polegające na tym, że bardzo duża wartość dodatnia może się stać bardzo dużą wartością ujemną, i odwrotnie. Przepełnienie zostanie zasygnalizowane zmianą stanu odpowiedniej flagi ACOVx.

2.42 Co to jest i jak realizowana adresacja z odwróceniem bitowym (BRA)?

Jest to sposób adresowania przeznaczony do przyspieszenia obliczeń programu transformat wykorzystujących sin() i cos() jako funkcje bazowe. BRA bazując na symetrii tych funkcji pozwala na przyspieszenie adresowania w buforach danych lub/i współczynników (zależnie od wariantu realizacji). Procesorowi należy przekazać informację o rozmiarze bufora zapisując ją lub liczbę z niej wynikającą do wskazanego rejestru. (w przypadku 'C5402 wpisujemy liczbę równą połowie rozmiaru bufora obsługiwanego tą adresacją do AR0). Poniżej w tabeli objaśnienie realizacji BRA.

	Licz (dec)	**Licz** (bin)	**BRA** (bin)	**BRA** (dec)	„Motylki"
wiersz	1	2	3	4	5
1	0	000	000	0	M0
2	1	001	100	4	
3	2	010	010	2	M2
4	3	011	110	6	
5	4	100	001	1	M1
6	5	101	101	5	
7	6	110	011	3	M3
8	7	111	111	7	

W trakcie adresacji powiększamy licznik a następnie odwracamy symetrycznie bity w zakresie potrzebnym do adresacji w buforze FFT (tutaj 8 lokacji zatem adresacja na 3 bitach). Połowę liczby wielkości bufora (#4) wpisujemy do AR0. Szczegóły można odnaleźć w przytoczonym dalej przykładzie programu do eksperymentowania z tym rodzajem adresowania.

Mechanizm;

- Odwrócenie bitów w wyznaczonym zakresie ==> do kol. 3
- Zwiększenie licznika z kol.1 o 1
- Odwrócenie bitów w wyznaczonym zakresie ==> do kol. 3
- Zwiększenie licznika z kol.1 o 1
- itd. ... do końca bufora

Proszę dostrzec, że zawartość kolumny 5 demonstruje „uczestników" kolejnych motylków ilustrujących obliczenia transformaty w podręcznikach. Wnikliwym załączam na końcu opisu zależności dla 8-mio punktowej FFT

Formuła programowa wymaga zainicjowania poza rejestrem adresującym również rejestru AR0. Poniżej fragment programu do demonstracji tego mechanizmu;

```
; Demonstracja BRA - Bit Reversal Addressing z użyciem listy rozkazów dla C54xx
; ============================================================
                .global  start, bufor_1, bufor_2, buf_test ; dla uwidocznienia w debugerze
                .mmregs          ; by umożliwić użycie predefiniowanych nazw rejestrów
```

```
;                ------    buforek danych do przeniesienia i sprawdzenia
;                          kolejności adresacji z odwróceniem bitowym

bufor_1          .sect "buf_test"
                 .word 00h        ; kolejne cyfry od 0 do 7 w kolejnych komorkach
                 .word 01h
                 .word 02h
                 .word 03h
                 .word 04h
                 .word 05h
                 .word 06h
                 .word 07h

;                ------    bufor_2 - miejsce na przepisanie w zmienionej
;                          kolejności po adresacji z odwróceniem bitowym
;                          Przetasowane dane znajdą się w kolejnych 8 komórkach

bufor_2          .usect   "bufor_2",8      ; rezerwacja miejsca dla d."przemieszanych"

                 .text
start
                 ;
; ----     test adresacji z odwroceniem bitowym
                 STM     #bufor_1,AR3  ; rejestr do adresacji BRA
                 STM     #bufor_2,AR1  ; rejestr docelowy
                 STM     #4,AR0        ; wielkosc bufora FFT/2
         NOP
                 LD      *AR3+0B,A     ;0 - pobieranie danej w adresacji BRA
                 STL     A,*AR1+       ; zachowanie danej w buforze wyniku
                 LD      *AR3+0B,A     ;1              -"-
                 STL     A,*AR1+
                 LD      *AR3+0B,A     ;2              -"-
                 STL     A,*AR1+
                 LD      *AR3+0B,A     ;3              -"-
                 STL     A,*AR1+
                 LD      *AR3+0B,A     ;4              -"-
                 STL     A,*AR1+
                 LD      *AR3+0B,A     ;5              -"-
                 STL     A,*AR1+
                 LD      *AR3+0B,A     ;6              -"-
                 STL     A,*AR1+
                 LD      *AR3+0B,A     ;7              -"-
                 STL     A,*AR1+
                 NOP                   ; bufor wynikowego tasowania pelen
```

2.43 Rozpisanie zależności dla FFT 8-mio punktowej N=8

$$X(m)=\sum_{n=0}^{N-1}x(n)*e^{-j2\pi*n*m/N}=\sum_{n=0}^{7}x(n)*e^{-j2\pi*n*m/8}=$$

$$X(m)=\sum_{n=0}^{(N/2)-1}x(2n)*e^{-j2\pi*(2n)*m/N}+\sum_{n=0}^{(N/2)-1}x(2n+1)*e^{-j2\pi(2n+1)/N}=$$

$$=\sum_{n=0}^{3}x(2n)*e^{-j2\pi*(2n)*m/8}+\sum_{n=0}^{3}x(2n+1)*e^{-j2\pi*(2n+1)*m/8}=$$

$$W_N^k=e^{-j2\pi k/N} \qquad W_N^2=e^{-j2\pi 2/N}=e^{-j2\pi/(N/2)} \qquad W_N^2=W_{N/2}^1$$

$$X(m)=\sum_{n=0}^{3}x(2n)*W_8^{2mn}+W_8^m*\sum_{n=0}^{3}x(2n+1)*W_8^{2mn}=\sum_{n=0}^{3}x(2n)*W_4^{mn}+W_8^m*\sum_{n=0}^{3}x(2n+1)*W_4^{mn}=$$

$$X(m)=\sum_{n=0}^{3}x(2n)*W_4^{mn}+W_8^m*\sum_{n=0}^{3}x(2n+1)*W_4^{mn}$$

$$X(m+4)=\sum_{n=0}^{3}x(2n)*W_4^{mn}-W_8^m*\sum_{n=0}^{3}x(2n+1)*W_4^{mn}$$

$$W_N^0=1\ W_N^{n/N}=-1 \qquad W_N^m W_{N/2}^m=W_N^{3m} \qquad W_N^m W_{N/2}^m W_{N/4}^m=W_N^{7m}$$

$$X(m)=x(0)+x(2)W_4^m+x(4)W_4^{2m}+x(6)W_4^{3m}+W_8^m\left[x(1)+x(3)W_4^m+x(5)W_4^{2m}+x(7)W_4^{3m}\right]$$

$$X(m)=x(0)+W_2^m x(4)+W_4^m x(2)+W_4^{3m}x(6)+W_8^m x(1)+W_8^{5m}x(5)+W_8^{3m}x(3)+W_8^{7m}x(7)$$

$$X(m)=x(0)+W_8^{4m}x(4)+W_8^{2m}x(2)+W_8^{6m}x(6)+W_8^m x(1)+W_8^{5m}x(5)+W_8^{3m}x(3)+W_8^{7m}x(7)$$

|- - - - M0 - - - - | |- - - - - - M2 - - - - - - | |- - - - - - M1 - - - - - | |- - - - - - M3 - - - - - - |

==========

2.44 Czym różni się linia magistrali od zwykłego połączenia punktów w układzie?

Magistrala to zespół linii oraz układów przyłączających służących do przesyłania sygnałów między połączonymi urządzeniami. Jest to zespół linii, zwykle równoległych, wraz ze zdefiniowanym sposobem przesyłania na niej danych. Charakteryzuje się przyłączeniem do każdej linii wielu wejść i wyjść układów. Połączenie wielu wyjść na wspólną linię tworzy specyficzne wymaganie unikania konfliktu w sterowaniu stanem linii. Dlatego należy zapewnić taki sposób podłączenia i sterowania, by o jej stanie decydowało tylko jedno wyjście a pozostałe nie wpływały na poziom sygnału na linii. Można to uzyskać używając z wyjściami trzystanowymi. Wyjścia takie poza logicznymi stanami 0 i 1 (L i H) mogą przyjmować stan trzeci - wysokiej impedancji (Z), czyli swego odłączenia od sterowania napięciem na linii. Stan ten odpowiada połączeniu wyjścia, do którego jest dołączona jest linia magistrali, dużymi rezystancjami do masy i do zasilania znikomo obciążającymi linię i sterujące nim aktywne wyjście innego układu. (patrz rys. 3.11 w książce Bruno Pilarda).
[BP, str. 39]

2.45 Opisz co to jest, co może zawierać i do czego służy stos?

Stos jest to fragment obszaru pamięci danych, na którym adresację zapewnia rejestr SP – wskaźnika stosu (Stack Pointer). Jest zorganizowany zgodnie z zasadą rejestru typu LIFO (Last In First Out), ostatnie zapisana dana jest pobierana jako pierwsza. Stąd pobieranie danych ze stosu jest realizowane jest w odwrotnej kolejności do ich składowania. W miarę odkładania kolejnych danych na stos jego obszar przyrasta on w stronę niższych adresów. Najpierw SP jest dekrementowany (– – SP) a potem dana kierowana jest do pamięci zgodnie z

aktualnym wskazaniem SP. Pobieranie danej ze stosu jest związane ze zwiększaniem SP – inkrementacją. Najpierw odczytujemy daną ze stosu (wskaźnik stosu SP wskazuje zawsze ostatnią zajętą pozycję na stosie) a następnie zwiększamy wskaźnik stosu (SP++) by wskazać kolejną daną do pobrania.
Stos służy głównie do zachowania i ochrony zawartości rejestrów procesora w trakcje realizacji procedur obsługi przerwań, przekazu parametrów do procedur (patrz język C), zachowuje adresy powrotów z procedur przywołanych do pracy rozkazem CALL oraz adresy kontynuacji pracy po ukończeniu ISR. Stos oddaje największe usługi w realizacji przerwania – są na nim zapisywane adres powrotu z przerwania i dane potrzebne do odtworzenia stanu procesora sprzed przerwania (context save/restore).
[BP, str. 122]

2.46 Dla procesora C5515 podaj co to jest stos, do czego służy, jak działa i co może zawierać?

Idea stosu zawiera się w określeniu –fragment pamięci danych, realizujący zasadę rejestru LIFO (Last In First Out - ostatnie zapisana dana jest pobierana jako pierwsza), na którym adresację zapewnia specjalny rejestr – wskaźnika stosu (Stack Pointer). W procesorach C55xx idea stosu jest rozbudowana. Procesor może dysponować dwoma stosami – stosem danych obsługiwanym rejestrem XSP i stosem systemowym obsługiwanym rejestrem XSSP. Oba te rejestry w procesorze C5515 mają rozmiar 23-bitów.

Rozmiar każdego z obu stosów ograniczony jest do 64K co wynika modyfikacji ich wskaźników jedynie na młodszych 16 bitach (SP[15:0] i SSP[15:0]). Starsza część rejestrów XSP i XSSP, to jest HSP[22:16] nie jest modyfikowana podczas operacji transferu na i ze stosu.

Oba stosy mogą pracować i być używane indywidualnie z możliwością dodatkowego wykorzystania lub nie szybkiego powrotu z procedur albo, jako stosy sprzężone (współbieżne) tworząc 32-bitowy stos jednak bez możliwości wykorzystania mechanizmu szybkiego powrotu z procedury. W przypadku pracy jako stosy współbieżne ich de- i in-krementacja są wykonywane równocześnie. Szybki powrót z procedury realizowany jest dzięki rejestrom RETA (Return Address Register) do zapamiętywania adresu powrotu i CFCT (Control-Flow Context Register) śladujący realizowane pętle i repetycje. Oba rejestry tworzą 32 bitowy obiekt dostępny do zapisu i odczytu.

Wybór sposobu działania stosu odbywa się poprzez odpowiednie ustawienie zawartości pierwszego bajtu wektora reset (procedury ISR dla Reset).
W miarę odkładania kolejnych danych na stosy ich obszar przyrasta w stronę niższych adresów – wskaźniki są dekrementowane. Pobieranie danych ze stosu jest związane ze zwiększaniem wskaźnika. Wskaźniki obu stosów wskazują zawsze ostatnią zajętą pozycję na stosie – szczyt stosu (TOS – Top of Stack).
Stos służy głównie do zachowania i ochrony zawartości rejestrów procesora w trakcje realizacji procedur obsługi przerwań, przekazu parametrów do procedur (patrz język C), zachowuje adresy powrotów z procedur, również rejestry RETA i CFCT. Stos oddaje największe usługi w realizacji przerwania – są na nim zapisywane adres powrotu z przerwania i dane potrzebne do odtworzenia stanu procesora sprzed przerwania (context save/restore).
[V3.x TechRef- str. 4.1]

2.47 Dla procesora C5515 podaj co to jest, co może zawierać, gdzie się znajduje i do czego służy pamięć podwójnego dostępu DARAM?

DARAM (Double Access RAM) pamięć dwukrotnego dostępu, czyli jest to pamięć zezwalająca na 2 dostępy w jednym cyklu procesora w każdym z bloków pamięci. Oznacza to, że przede wszystkim procesor może pobrać dwa operandy z tego samego bloku.

Pamięć ta mieści się we wnętrzu procesora i przeznaczona jest głównie na dane. W procesorach C5515 ma ona rozmiar 64K. Może być również wykorzystywana jako pamięć programu ale nie jest to jej efektywne wykorzystanie i trzeba zwrócić uwagę na połączenie obszarów logicznych pamięci programu i danych – czyli powrót w części obszaru mapy pamięci do architektury von Neumana.
Część rozkazów może być efektywnie wykonywana tylko wtedy, gdy ich operandy rozmieszczone są w DARAM. Operandy niektórych rozkazów muszą być umieszczone w tej pamięci, bo inaczej działanie na nich jest nieefektywne a czasami wręcz niemożliwe (patrz rozkazy z grupy MAC, rozkazy specjalistyczne czy równoległe).
Jej zastosowania wynikają z własności – powinna zawierać dane dla najszybszych fragmentów działającego programu, używającego najbardziej złożonych i zrównoleglonych operacji.
[C5515_Data]

2.48 Dla procesora C5515 podaj na czym polega operacja nasycania i do czego służy?

Operacja nasycania polega na wprowadzeniu ograniczenia wyniku operacji w akumulatorze i zapobieganiu błędów wynikających działania modulo-40 rejestru akumulatora. Sposób realizacji nasycania uzależniony jest od stanu bitu M40, który decyduje;
= czy nasycanie odbędzie się na poziomie bitu 32 / M40=0 (dla 32-bitowej reprezentacji U2 liczby – maksymalna dodatnia (0x00 7FFF FFFFF) / maksymalna ujemna (0xFF 8000 0000),
= czy na poziomie bitu 40 / M40=1 (dla 40-bitowej reprezentacji U2 liczby – maksymalna dodatnia (0x7F FFFF FFFFF) / maksymalna ujemna (0x80.0000 0000)
Operacja nasycania może być realizowana na kilka sposobów

- Nasycanie zawartości akumulatora, który przekroczył zakres reprezentacji 32 bitowej rozkazem SAT
- Nasycanie zawartości akumulatora przy operacjach SHIFT, bez specjalnego polecenia – w zależności od ustawienia M40 i SXMD
- Nasycanie wyniku odsyłanego do pamięci z akumulatora ale bez nasycania zawartości samego akumulatora – Saturation on Store. Uruchamiane jest ustawieniem bitu SST w rejestrze statusowym ST3_55. W tej metodzie sposób nasycania zależy od włączonego lub nie trybu rozszerzenia znakowego (Sign Extention Mode).
  Dla włączonego – SXMD=1 – ograniczenie następuje na liczbach maksymalnej (0x00 7FFF FFFFF) / minimalnej (0x00 8000 0000).
  Dla wyłączonego – SXMD=0 – ograniczenie następuje na liczbach maksymalnej (0x00 FFFF FFFFF) / minimalnej (0x00 0000 0000).
  Odsyłając do pamięci „nasyconą" wartość nie nasycamy zawartości akumulatora!

Metody te służą do usprawniania wyniku dla różnych warunków obliczeń. Upraszczając nieco, Przeciwdziałają bezzasadnej zmianie znaku liczby przez „przekręcenie się" rejestru / licznika co może być równoznaczne przeskokowi odpowiadającego sygnału analogowego z maksymalnej wartości ujemnej do maksymalnej wartości dodatniej.
Nasycanie wyników jest odpowiednikiem ograniczenia sygnału na napięciu zasilania w układach wzmacniaczy analogowych.

2.49 Co trzeba zapewnić (co musi być przygotowane) dla prawidłowej obsługi przerwania w procesorze C5515?

Dla prawidłowego działania procedury obsługi przerwania należy;

- Opracować program obsługi przerwania (ISR)
- Przygotować i rozmieścić wektory procedur ISR w tablicy wektorów przerwań – przygotować odpowiedni wektor (.ivec *isr)*.
- Ustawić globalną maskę przerwań - INTM,
- Ustawić odpowiedni bit indywidualnej maski danego przerwania w rejestrze masek przerwań – IMR,
- Ustawić wartość wskaźników stosu – XSP, XSSP (zapewnić miejsce na stosy)

- Przygotować zawartość całej tablicy wektorów przerwań a szczególnie IVPD, wskazujący na początek naszej tablicy wektorów, a w razie potrzeby zainicjować używanie tej tablicy rozkazem RESET wykonanym po ustaleniu zawartości IVPD, IVPH (dla wektora RESET konieczne jest zawarcie w wektorze definicji sposobu pracy stosu
  label: **.ivec** [*address*[, *stack mode*]]

[C55xx_AssLangTools, str. 174]

2.50 Dla procesora C5515 opisz krótko tryb adresacji pośredniej, jego przeznaczenia i podaj przykłady rozkazów?

Adresacja pośrednia(indirect) –
Ogólnie to tryb adresacji prostej, gdzie pojedyncze słowo binarne zawiera kod rozkazu i wskazuje rejestr – zawierający adres operandu w pamięci danych. Adresowanie pośrednie realizuje ideę wskaźnika.
W procesorze DSP z rodziny C55xx jest to adresacja z użyciem rejestrów adresujących AR0-7, oznaczana *ARn, lub rejestr CDP (Computed Data Pointer) z oznaczeniem *CDP. Jest to sposób adresacji najbardziej wszechstronny, sprawny, z gamą modyfikacji zawartości rejestrów dla sprawnej obsługi tablic, filtrów (operacje arytmetyczne na rejestrach AR) i transformacji (modyfikacja rejestrów, BRA).
Zasięg adresacji w tym trybie wynika z rozmiaru rejestru XARx – 23 bity – co pozwala adresować całą przestrzeń pamięci procesora (8 MBajtów).

Przykład:
```
MOV    *AR1,AC1          ; zawartość komórki adresowanej przez AR1 jest wpisywana
                         ; do akumulatora AC1
MOV    #0x1000,*AR5-     ; stałą 1000h jest  zapisywana w komórce pamięci danych
                         ; adresowanej przez zawartość rejestru AR5.
                         ; potem AR5 jest dekrementowany
                         ; #0x1000  => DRAM(AR5)
                         ; AR5 – 1 => AR5
```

[C55xx_Mnem_InSRG, str. 3.6]

2.51 Z jakiego punktu przestrzeni adresowej uruchamiany jest program po RESET sprzętowym a z jakiego po RESET programowym w C5515?

Punkt startu programu po sprzętowym RESET jest niezmienny dla danego procesora i dla rodziny C5515 wynosi 0xFFFF00. Programista może przełączyć procesor do odczytywania tablicy wektorów przerwań z innego miejsca w pamięci programu przez zmianę zawartości IVPD i wykonanie po tym programowego RESET. Wówczas po tym rozkazie zostanie pobrany z początku tablicy wektorów przerwań (IVPD * 256) wektor procedury reset - adres startowy programu.

2.52 Czym różni się realizacja pętli programowej od pętli wykonanej dzięki trybowi repetycji w procesorach DSP (np. C5515)?

Tryb repetycji jest formą realizowania pętli programowych. Może je wykonywać dwoma sposobami
- poprzez powtarzanie pojedynczego, zwykle złożonego rozkazu
- przez powtarzanie kilku rozkazów w bloku.

W tym trybie procesor obsługuje sprzętowo licznik powtórzeń pętli (poprzez inicjowany wcześniej rejestr repetycji RC lub BRC dla bloku) oraz skok na początek pętli (za sprawą dwóch rejestrów, RSA – początku bloku i REA – końca bloku)

Pętla programowa jest mniej efektywna czasowo od tej z trybu repetycji gdyż musi programowo sprawdzać licznik powtórzeń i realizować skok i te rozkazy wymagają czasu na wykonanie.
Różnic jest jednak więcej;

- przerwania – pętla programowa dopuszcza przerwanie (jeśli tylko są odblokowane) zaś repetycja pojedynczego rozkazu nie dopuszcza przerwań, nawet gdy są odblokowane. Muszą czekać na zakończenie repetycji. Natomiast repetycja bloku może dopuścić przerwanie.
- Zagłębienie pętli – jest możliwe dla pętli programowej, niemożliwe dla repetycji pojedynczego rozkazu a ograniczone dla repetycji bloku

[V3.x CPU]

2.53 Czym różnią się rozkazy INTR n od TRAP n?

Oba rozkazy przywołują wykonanie n-tej procedury obsługi przerwania bez względu na stan maski danego przerwania. Różnica polega na tym, że na początku procedury przywoływanej rozkazem INTR następuje zablokowanie przerwań. Powoduje to zablokowanie systemu przerwań i nie zauważanie zgłoszonych przerwań. ISR przywołana rozkazem TRAP nie blokuje systemu przerwań.
[V3.x InstSet_RG]

2.54 Dla procesora C5515 podaj, na czym polega operacja zaokrąglania, do czego służy i jak się ją uaktywnia?

Operacja zaokrąglania wyniku operacji w akumulatorze polega na uwzględnieniu w starczej części akumulatora ACH części młodszej ACL w dwojaki sposób. Albo zaokrąglamy stronę nieskończoności albo w stronę najbliższej wartości. O sposobie zaokrąglania decyduje bit RDM (Rounding Mode Bit), 10 bit z rejestru statusowego (ST2_55.10)
Jeśli RDM=0 i prowokowane jest zaokrąglanie wówczas do wskazanego akumulatora (lub dwóch akumulatorów równocześnie) dodawana jest wartość `0x8000` po wykonaniu operacji rozkazu.
Jeśli zaś RDM=1 to wywoływane zaokrąglanie odbędzie się w zależności od zawartości młodszej części akumulatora. I tak jeśli `(8000h < bity(15-0) < 10000h)`
to do 40-bitowego akumulatora ACx dodajemy `8000h, tj. ACx=(ACx + 0x8000)`
Jeśli zaś `( bit(15-0) == 8000h) i bit(16) == 1)`
to analogicznie `ACx=(ACx + 0x8000)`
Po operacji zaokrąglania młodsza część akumulatora ACL jest zerowana, (ACL=0)
W ten sposób w odbieranym wyniku z 16-to bitowej starszej części akumulatora można uwzględnić część młodszą, jeśli jej wartość jest większa lub równa ½ LSB starszej części tego akumulatora.
Zaokrąglanie służy „usprawnianiu wyniku", ale musi być stosowane z ostrożnością i rozwagą dla uniknięcia kumulacji błędów obliczeń i zwykle dotyczy ostatniej operacji ciągu obliczeń.
Zaokrąglanie uruchamiane może być normalnym rozkazem ROUND albo skojarzone z rozkazem operacji arytmetycznej dzięki rozszerzaniu mnemonika rozkazu o końcową opcjonalną literę „R" (np. MAC[R], MAS[R], MPY[R], ADD[R], ...).
[V3.x InstSet_RG, str. 5.528]

2.55 Które rejestry procesora w trakcie obsługi przerwania są zachowywane automatycznie a które musi zachować procedura ISR?

Automatycznie mogą zostać zachowywane na stosie; adres powrotu i rejestr kontekstu realizacji pętli repetycji (razem tworzą 32-bitowe słowo), rejestr statusu debugowania DBSTAT i 3 rejestry statusowe ST0_55, ST1_55 i ST2_55.
Procedura ISR musi sama zachować na stosie te rejestry, których będzie używała w trakcie swej pracy, czyli będzie zmieniała ich zawartość.
[C55xx_Mnem_InSRG, str. 4.8]

2.56 Dla procesora C'5515 opisz krótko tryb adresacji bezpośredniej, jego przeznaczenie i podaj przykłady rozkazów?

Adresacja bezpośrednia w procesorach rodziny C5515 realizuje adresowanie stronicowane składając 23 bitowy adres XDP z 7-mio bitowego rejestru DPH i 16-to bitowego kodu będącego sumą 7-miu bitów z kodu rozkazu i zawartości rejestru DP lub rejestru wskaźnika stosu SP. Wybór rejestru do sumowania wykonywany jest zawartością bitu CPL z rejestru statusowego ST1_55. Dla CPL=0 do tworzenia adresu wykorzystywany jest rejestr DP a dla CPL=1 rejestr SP. W ten sposób rejestr XDP wskazuje jedną z 128 stron głównych a DP wskazuje stronę lokalną na stronie głównej. SP wskazuje TOS (szczyt stosu) tworząc w trybie adresacji bezpośredniej bardzo efektywną metodę dostępu do przekazywanych przez stos obiektów.
Jest to użyteczna adresacja o zwartym rozkazie zajmującym jedno słowo w pamięci programu, potrzebującym jednego cyklu na wykonanie w kolejce procesora – rozliczanym w wykonywaniu programu jednym cyklem procesora.
Ograniczeniem tego trybu jest mały zasięg adresacji obejmujący tylko jedną, 128 elementową stronę danych.
Np. MOV @x, AC1 ; załaduj wartość spod adresu „x" z aktualnej strony danych (lub ze stosu) do akumulatora AC1
Tryb adresacji bezpośredniej ma zastosowanie również do dostępu do portów we/wy. Wykorzystuje się wówczas jako rejestr strony 9-cio bitowy rejestr PDP (Port Data Page) złożony z 7 bitami z kodu rozkazu. Wymaga to użycia w rozkazie kwalifikatora port().
Bezpośrednia adresacja używana jest również w rozkazach testowania i manipulowania bitami w głównych rejestrach procesora (ACx, ARx, Tx). Kwalifikatorem @bitoffset określamy pozycje bitu w rejestrze.
[V3.x InstSet_RG, str. 3.4]

2.57 Czym różnią się od siebie procedura obsługi przerwania od dowolnej innej procedury programowej procesora C5515?

ISR	Procedura programowa
Może „wtrącać się" w program i inne procedury	Tylko wykonywana przez odwołanie z programu
Ma ustalone punkty startu procedur poprzez IPTR	Umieszczana w dowolnym punkcie programu
Może być również uruchamiana z programu `INTR n` i `TRAP n`	Uruchamiana tylko z programu przez rozkaz `CALL proc`
Powiązane ze zdarzeniami sprzętowymi wewn. i zewn.	Bez mechanizmów powiązania ze sprzętem
Automatycznie blokuje przerwania i wymaga specjalnej zgody na ich dopuszczenie	analogiczny status przerwań jak cały program
Wymaga inicjatywy dla odblokowania przerwań	Wymaga inicjatywy dla zablokowania przerwań
Różni je status ważności - priorytet	Mają równy status ważności – bez priorytetów
Dla dopuszczenia działania wymaga ustawienia odpowiednich flag	Nie zależy od flag

2.58 Co to są sekcje programu w C'5515, jakie są ich rodzaje i do czego służą?

Sekcje to fragmenty programu, z których każda zawiera jednorodne obiekty; kod, stałe, zmienne lub układy we/wy. Wyznaczane są w programie za pośrednictwem dyrektyw oznaczających początek nowej sekcji.
Sekcje mogą być inicjowane (o znanej zawartości w chwili tworzenia np. zawierające kod programu czy stałe) lub nieinicjowane (tylko rezerwujące miejsce na zmienne lub stałe). Mogą być opisywane własnymi nazwami – nazwane lub nie nazwane, bez określonej nazwy.

Sekcje wprowadzane są dla ułatwienia rozmieszczania segmentów programu w pamięciach procesora. Rozmieszczenie to jest wykonywane przez Linker w oparciu o jego zbiór konfiguracyjny będącym zapisem rozmieszczenia zasobów sprzętowych i wymagań programisty na kierowanie poszczególnych sekcji do istniejących zasobów.
Linker realizuje polecenia zbioru konfiguracyjnego oraz polecenia niektórych dyrektyw z programu źródłowego. Dodatkowo zmierza do łączenia sekcji o takich samych nazwach we wspólne obszary

2.59 Czym różni się przerwanie sprzętowe od przerwania programowego?

Przerwanie sprzętowe (ang. Hardware Interrupt), powodowane jest zdarzeniem w zasobach sprzętowych procesora wewnętrznych (przepełnienie licznika, koniec transmisji danych, koniec przetwarzania wewnętrznego przetwornika A/C itp.), lub zewnętrznych za sprawą sygnałów doprowadzony do wejść przerwań (INT0, INT1, … INTn, NMI, RESET).
Cechą charakterystyczną przerwań sprzętowych jest, że poza NMI i RESET dysponują swoimi flagami rejestrującymi zgłoszenie przerwania – żądanie obsługi (w rejestrze IFR) i indywidualnymi maskami przerwań (w rejestrze IMR) pozwalającymi na ich indywidualne blokowanie obsługi, niezależnie od możliwości ogólnego blokowania całego systemu przerwań.
Cechą charakterystyczną przerwań sprzętowych są ich priorytety – przypisane poziomy ważności, wyznaczające kolejność wyboru do obsługi w przypadku równoczesnego zgłoszenia żądania obsługi dwóch lub więcej.
Ze względu na swe powiązania sprzętowe służą do synchronizacji programu z asynchronicznymi względem niego zdarzeniami.
Przerwania programowe wywoływane są z programu rozkazami `INTR n` lub `TRAP n`
Oba rozkazy przekierowują sterowanie z programu do „n-tej" procedury obsługi przerwania w tablicy wektorów przerwań. Mają zdolność sprowokowania każdej z procedur umieszczonych w tablicy wektorów przerwań. Rozkaz `INTR n` dodatkowo blokuje system przerwań ustawiając globalną maskę przerwań INTM. Jako że wywoływane są z programu więc nie mają powiązań z zdarzeniami sprzętowymi (ani zewnętrznymi ani wewnętrznymi) zatem ze swej istoty nie nadają się do synchronizacji z zewnętrznym światem biegnącym niezależnie od przebiegu programu.
[BP, str. 153]

2.60 Dla procesora 'C5515 opisz krótko tryb adresacji natychmiastowej, jego przeznaczenie i podaj przykłady rozkazów?

Adresacja natychmiastowa (immediate) –
Jest to tryb adresacji w którym używany operand, jego wartość, jest zawarty w rozkazie. Rozkaz, w zależności od wartości operandu (danej) może mieć rozmiar jednego lub dwóch słów. Operand zawarty w rozkazie jest przechowywany w pamięci programu wraz z kodem rozkazu i jest do natychmiastowego użycia bez konieczności sięgania do pamięci danych.
Ten tryb adresacji wykorzystywany jest zwykle do inicjalizacji rejestrów.
Przykład: **`MOV #100,ACx`** ; wartość 10d jest wpisywana do akumulatora A
; 0x100 => ACx

2.61 Co to jest zbiór konfiguracyjny linkera w 'C5515 i do czego służy?

Zbiór konfiguracyjny linkera to zabiór tekstowy, niezbędny dla prawidłowego przebiegu procesu linkowania programu – prawidłowego działania programu Linker.
Zawiera w sobie wykaz zbiorów wejściowych jakie Linker będzie brał do obróbki wraz z kolejnością ich pobierania, wykaz zbiorów wyjściowych wytwarzanych przez Linker, zestaw opcji konfigurujących pracę Linkera, zapis rozmieszczenia zasobów pamięci programu, danych i portów we/wy (adresy początkowe i wielkość obszaru) oraz skierowania sekcji programu do odpowiednich obszarów pamięci. Zatem zbiór ten określa rozmieszczenie zdefiniowanych w asemblerze bloków (fragmentów) programu w fizycznej mapie pamięci jaką dysponuje system.

Zwykle zbiór ten opatrzony jest rozszerzeniem (*.cmd)

2.62 Objaśnij jakie operacje wykona rozkaz: PSH *AR2- procesora ‘C5515?

Bardzo skrótowo to najpierw zmniejsza wskaźnik stosu SP o jeden, odsyła daną z adresu wskazywanego w rejestrze AR2 na stos pod adres zawarty w SP, zmniejsza zawartość AR2 o 1.

```
(SP) - 1  → SP          ;dekrementacja wskaźnika stosu
(AR2) → TOS(SP)         ;przepisanie zawartości komórki pamięci adresowanej przez AR2 na
                        ;szczyt stosu (wskazuje go SP
AR2 – 1 → AR2           ;dekrementacja AR2
```

2.63 Skąd linker wie jakie sekcje ma łączyć ze sobą w ‘C5402 a jakie nie?

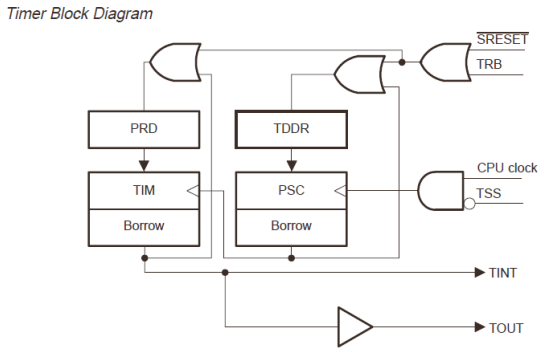
Sekcje są tworzone w programie źródłowym za pomocą dyrektyw. Mogą być opatrzone nazwą (nazwane) lub nie (nienazwane). Linker z natury działania będzie łączył ze sobą sekcje o takich samych nazwach. Dodatkowo można za pośrednictwem skierowań sekcji w zbiorze konfiguracyjnym linkera zlecić linkerowi sposób i kolejność umieszczania sekcji w pamięci danych i programu, kierując sekcje do odpowiednich obszarów pamięci zdefiniowanych również w tym zbiorze konfiguracyjnym.

2.64 Flaga a maska

Zarówno flaga jak i maska reprezentowane są stanem przerzutnika w rejestrze. Są zatem obiektami o rozmiarze jednego bitu. Wyróżniają je funkcje.
Maska jest zazwyczaj bitem blokującym lub dopuszczającym jakiś mechanizm, zdarzenie, np. obsługę przerwania a flaga jest bitem rejestrującym jakiś fakt, zgłaszającym jakąś potrzebę, np. obsługi przerwania, czy ustalającym jakiś fakt, np. Poziom sygnału wyjściowego na wyjściu XF (eXternal Flag).
Flaga może być ustawiana sprzętowo za sprawą wystąpienia zewnętrznego sygnału (jak przerwania), czy jakiegoś stanu (np. przekroczenia zakresu w akumulatorze A – OVA) lub może być ustawiana programowo – rozkazem. Niektóre flagi mogą być również kasowane (przestawiane) programowo, a są również takie, które kasuje się tylko za sprawą podjęcia jakiegoś działania (np. flagi przerwań) lub wykonania specyficznego rozkazu (np. flagę OVA można skasować TYLKO wykonując skok warunkowy sprawdzający warunek OVA).
Bity masek mogą być ustawiane i kasowane programowo a są maski ustawiane również za sprawą zdarzeń. Dla przykładu maska globalna przerwań – INTM ustawiana jest sprzętowo przy rozpoczynaniu obsługi przerwania ale i po RESET.
Zarówno flagi jak i maski mogą być elementami rejestrów statusowych (ST0, ST1, PMST) lub rejestrów specjalistycznych czy konfiguracyjnych np. IFR czy IMR.

2.65 Co to jest TIMER w systemie DSP

Mianem TIMER określa się w DSP podobnie jak w technice procesorowej zespół układów działających we współpracy z licznikami impulsów w celu zliczania, odmierzania czasu czy generacji impulsów.



Podstawowa idea pracy sprowadza się do wstępnego załadowania rejestru licznika wartością określającą wymagany czas dekrementowania sygnałem taktującym do stanu 00 powodującego wygenerowanie sygnału i ewentualne odświeżenie początkowego stanu licznika dla umożliwienia powtórzenia całej operacji.
Konkretny sposób obsługi, powiązanie ze źródłami sygnałów taktujących, długość licznika, sposób sygnalizacji, itd. zależy od konkretnej implementacji i procesora.
Przeznaczenie układów TIMER patrz pytanie Do czego służy w procesorach DSP zegar (timer)?.

2.66 Objaśnij co może ułatwiać w C5515 DSP wykorzystanie rozkazu FIRS– uzasadnij?

Rozkaz umożliwia uproszczenie, przyspieszenie obliczania symetrycznego filtru FIR. Filtr musi mieć symetryczny układ współczynników by możliwe było zsumowanie wartości próbek sygnału przyporządkowanych współczynnikom o takich samych wartościach/

```
FIRS *AR3+, *AR4+, COEFFS   ;COEFFS to adres pmad (p.programu)
                            ;COEFFS → PAR
                            ;While (RC) różne od 0
                            ;      (B)+(A(32-16))*(PAR) → B
                            ;      ((AR3)+(AR4)) << 16 → A
                            ;      (PAR)+1 → PAR
                            ;       AR3+1 → AR3
                            ;       AR4+1 → AR4
                            ;      (RC)-1 → RC
                            ;zależy od bitów SXMD, FRCT i OVM
                            ;ustala bity C, OVA i OVB
```

2.67 Architektura Von-Neumana

Architektura ta opisuje system z jednolitą przestrzenia adresową, obsługiwaną przez jedną szynę, wspólna dla danych i kodów programu. W tej architekturze podział na obszar pamięci na dane i program jest umowny. Zaletą tej architektury jest łatwość programowania, gdyż dostęp do danych, programu i urządzeń we/wy odbywa się przy użyciu zunifikowanych rozkazów wykorzystujących te same tryby adresowania. Nie jest potrzebne wprowadzenie specjalnych rozkazów pozwalających na przepływ danych pomiędzy pamięcią danych i programu. Niestety dysponując tylko to dysponując tylko jedną magistralą nie możemy wykonywać równoległych przesłań, stąd wszystkie operacje odczytu i zapisu muszą odbywać jedna po drugiej. Wszystkie operacje wykonywane są sekwencyjnie, jedna po drugiej. Nie ma możliwości zrównoleglenia operacji i przyspieszenia wykonywania programu.

2.68 Architektura Harwardzka

Głównym elementem tej architektury jest wyraźne i stałe rozdzielenie obszarów pamięci danych i programu oraz dwie oddzielne szyny magistral do transportu danych i kodów programu. Dzięki temu można wykonywać równolegle operacje pobierania kodu i danych (operandów rozkazów) a dzięki specjalnej organizacji zasobów również wykonywać równolegle inne fazy rozkazów organizując

kolejkę przetwarzania nakładkowego. Kolejka, w której kilka kolejnych rozkazów z pamięci kodu jest „nasunięte na sienie" Przyspiesza realizację programu. Kolejka NIE SKRACA cyklu rozkazowego (rozkazy nadal wymagają takiej samej liczby faz – cykli procesora, – ale przez równoległe wykonanie kilku faz rozkazów skraca sumaryczny czas wykonywania programu. Rozdzielenie obszarów pamięci programu i danych wymaga uzupełnienia sygnałów sterujących (select) wyboru odpowiedniego obszaru pamięci lub portów we/wy.
Omówienie organizacji magistrali patrz pytanie; O czym mówimy używając określenia „magistrala danych" czy „magistrala programu"?

2.69 Udoskonalenie architektury harvardzkiej w DSP.

Architektura Harward została w procesorach DSP skorygowana w ten sposób, że zwiększono liczbę magistral danych z jednej do trzech. Umożliwia to równoległe pobieranie z pamięci danych do dwóch argumentów dla rozkazów obliczeniowych (np. MAC) i odsyłanie wyniku operacji do pamięci danych. Zwiększa to wydajność procesora, ale i jego złożoność. Uzyskany dla CPU procesora dostęp do kilku obszarów jednocześnie (wsparty dzięki zastosowaniu pamięci DARAM) zwiększa przepustowość transferu danych między pamięcią a CPU.

2.70 DEBUGGER

(ang. Odpluskwiacz, {de-przeciwieństwo, bug-robak} - debugger) jest to komputerowy program pozwalający testować oraz „debugować" inne programy. Sam debugging jest pojęciem związanym z procesem metodologii szukania i redukowania liczby błędów lub defektów, w wymienionym programie komputerowym czy procesora, dzięki czemu dany program będzie się zachowywać zgodnie z oczekiwaniem. „Debugowanie" bywa trudniejsze, kiedy różne podprogramy są ściśle powiązane, jedne zmiany mogą powodować błędy w innym podprogramie, lub podsystemie. Ważna jest świadomość, że sam debugger nie usuwa błędów a jedynie daje użytkownikowi możliwość zatrzymania programu w wymaganych miejscach i kontroli zawartości rejestrów czy pamięci w celu dokonania oceny prawidłowej kolejności pracy czy obliczeń. Debugger nie ma żadnych automatycznych mechanizmów wykrywania błędów programu. To zadanie spoczywa zawsze na użytkowniku programu.
Debugging można prowadzić na różnych poziomach w zależności od tego, w jakim języku napisany został testowany program, czy testowaniu poddawany jest kod asemblerowy (ze znacznie precyzyjniejszą, ale i żmudniejszą kontrolą zależności czasowych pracy programu) czy kod programu w języku wyższego poziomu.

2.71 EMULATOR a DEBUGER

Emulator to zestaw; urządzenie + oprogramowanie + zasoby na procesorze, służący do połączenia komputera host z procesorem DSP. Wykorzystuje możliwości standardu komunikacji JTAG i zawarte wewnątrz procesora specjalne zasoby dla umożliwienia kontroli nad pracą procesora DSP i uruchamiania programu w procesorze. Zwykle współpracuje z programowym środowiskiem generowania i uruchamiania programu na procesorze CCS (Code Composer Studio) w tym programem debugera.
Debugger to program działający na komputerze host (w ścisłej współpracy z programem na procesorze DSP) pozwalający na podgląd zasobów podłączonego za pomocą emulatora procesora DSP (szczególnie rejestrów i pamięci), ustawianie w uruchamianym programie punktów zatrzymania (pułapek – breakpoints) i punktów testowania (probepoints) i wiązanie z nimi różnych operacji. Wszystko to dla potrzeb testowania i korygowania działania badanego programu, usuwania jego błędów (debugowania) programu na procesorze docelowym DSP, korzystając zarówno z programu na komputerze host, zasobów wewnętrznych procesora DSP dedykowanych dla debugingu jak i z fizycznego połączenia między DSP a komputerem host.
Zatem obydwa tworzą łącznie środowisko sprzętowego uruchamiania programu na procesorze docelowym DSP korzystając zarówno z programu na komputerze host, zasobów wewnętrznych procesora oraz z połączenia fizycznego pomiędzy procesorem DSP a komputerem.
W przypadku procesorów produkcji Texas Instruments pracę tego zestawu usprawnia użycie systemu czasu rzeczywistego DSP/BIOS działającego na procesorze DSP.

2.72 LINKER

Linker jest programem, który łaczy pliki składające się na program po ich uprzedniej kompilacji / asemblacji do formatu plików *.obj. Wykonuje to posługując się poleceniami zawartymi w linii poleceń lub w pliku konfiguracyjnym linkera. Generuje docelowy zbiór z programem *.out do wykonania na procesorze DSP. W trakcie swojej pracy łączy sekcje o takich samych nazwach i rozmieszcza sekcje w obszarach pamięci wskazanych w zbiorze konfiguracyjnym linkera *.cmd.
Zbiór konfiguracyjny linkera - *.cmd (linker configuration scripts file) – jest niezbędny do prawidłowego zbudowania kodu; definiuje gdzie w pamięci programu umieszczone są fragmenty kodu, bloki pamięci; opisuje przyporządkowanie plików wejściowych wyjściowym, definiuje alokacje pamięci. Linker może również generować szereg użytecznych w trakcie uruchamiania zbiorów, opisujących np. rozmieszczenie obiektów i procedur programu (zbiór *.map) czy zbiór pełnego listingu programu (*.lst) zawierający zapis programu, adresy, wykonywalny kod, komentarze.
[BP, str. 79], [C54x_CPU_Priph_spru131g.pdf; str. B.1]

2.73 Architektura harwardzka a super harwardzka.

Główne cechy architektury Harward opisane zostały w p. 2.68.
Skorygowana architektura Harward, jaką posługuje się Texas Instruments przwiduje zwiększenie liczby magistral danych do 3 i nadanie im pewnej specjalizacji. Dwie magistrale C i D ukierunkowane są na równoległo odczyt operandów dla wykonania rozkazu. Zaś trzecia z nich – E, jest przeznaczona do odesłania wyników obliczeń do pamięci lub układów We/Wy.
Hasło „Super Harward" pojawia się w kontekście procesorów SHARC firmy Analog Devices (Super Harvard ARchitecture Computer). Architekturę tę można określić jako Harward o wydłużonym słowie rozkazowym. Terminologia ta jest stosowana trochę marketingowo i nie należy jej mylić z określeniem SuperH SH) firmy Hitachi.

2.74 Co to jest rozszerzenie znakowe, czemu służy itd?

Procesory sygnałowe dla zachowania odpowiedniej precyzji obliczeń zwykle dysponują akumulatorami, co najmniej dwukrotnie większymi od rozmiaru słowa, którym pracują. W przypadku rodziny procesorów 'C5000 pracującej na słowie 16-to bitowym akumulatory mają rozmiar 40-to bitowy. Rozszerzenie znakowe to mechanizm pozwalający procesorowi w takiej sytuacji na zachowanie znaku danej ładowanej do większego rejestru. Operacja ta realizowana jest automatycznie i może być włączana za pomocą bitu SXMD – Sign eXtension Mode – umieszczonego w rejestrze statusowym ST1.
[BP, str. 99]

2.75 Co to są dekodery adresów, jaka jest ich rola w systemie DSP?

Dekodery adresów są to cyfrowe układy kombinacyjne podłączane wejściami do linii adresowych procesora a wyjściem do odpowiednich układów pamięci lub portów we/wy, które mają mają aktywować do komunikacji z procesorem, gdy na liniach adresowych pojawi się adres odpowiadający układowi, które obsługują. Służą do aktywowania właściwego układu pamięci lub portu urządzenia zewnętrznego na podstawie generowanego przez mikroprocesor słowa adresowego i odpowiednich sygnałów sterujących.
[BP, str. 27]

2.76 Co to jest i do czego służy emulator (ICE) procesora DSP?

Emulator to urządzenie sprzętowo-programowe. Jego zadaniem jest zapewnienie połączenia pomiędzy narzędziami programowymi na komputerze host a zasobami wewnętrznymi procesora DSP. Celem tego połączeni jest umożliwić programowi Debugera kontrolę nad pracą procesora DSP a co za tym idzie uruchamianie i testowanie programu. W swej pracy wykorzystuje połączenie JTAG (standard

Boundary Scan) wraz z mechanizmem wymiany danych w czasie rzeczywistym RTDX (Real Time Data eXchange) oraz wewnętrzne, przygotowane specjalnie do wsparcia debugowania zasoby w strukturze procesora DSP (zwykle rejestry, komparatory, w bardziej rozbudowanych pamięć przechwytu). Po stronie komputera host emulator jest obsługiwany przez sterowniki zwykle będące elementem środowiska developerskiego do przygotowania, generowania i uruchamiania kodu (w przypadku produktów Texas Instruments CCS – Code Composer Studio), którego elementem jest program uruchamiania i testowania – debuger i inne programy optymalizacyjne i diagnostyczne jak np. profiler. Ich sprawne i efektywne działanie jest w znacznym stopniu wyznaczane jakością i szybkością działania emulatora.
Emulator jest w praktyce jedynym narzędziem pozwalającym na wgląd w zasoby procesora i śledzenie ich pracy w czasie rzeczywistym, gdy procesor pracuje na pełnej szybkości.
Wykorzystanie emulatora przekłada się na ułatwienie prac uruchomieniowych oprogramowania procesora, poprawę możliwości diagnostycznych błędów programu działającego z pełną szybkością i z podłączonymi peryferiami a dzięki temu możność sprawdzenia programu w prawdziwych warunkach i szybsze przygotowanie produktu.
[BP, str. 52]

2.77 Mnemonik

(skrót mnemotechniczny) - symbol (słowo) utworzony zgodnie z zasadami mnemotechniki, tzn. w taki sposób, aby forma zapisu była pomocna w zapamiętaniu, do jakiej operacji przypisane jest dane słowo-kod. Mnemonik składa się z kilku liter będących skrótem słów z języka angielskiego oznaczających daną czynność procesora. Listy rozkazów procesorów, w których skład wchodzą mnemoniki różnią się w zależności o typu procesora. Niektóre z mnemoników, oznaczające takie operacje jak dodawanie, odejmowanie, mnożenie czy dzielenie są jednakowe dla wszystkich procesorów. Za zamianę (tłumaczenie) zapisu programu z użyciem mnemoników na język wewnętrzny (kod maszynowy) "zrozumiały" dla procesorów odpowiedzialne są specjalnie przeznaczone do tego programy zwane Asemblerami.

Mnemonik	**Nazwa**
MOV	przesuń
ADD	dodaj
CALL	przywołaj procedurę
B	skocz

[C54x_Mnem_Instr_Set_spru172c.pdf; str. 2.1], [BP, str. 82]

2.78 Co to są sekcje programu, co mogą zawierać, jakie są ich rodzaje i do czego służą w `C5515?

Sekcje to fragmenty programu zawierające jednorodne obiekty; kod, stałe, zmienne lub układy we/wy. Są one zdefiniowane za pomocą dyrektyw w zbiorach źródłowych programu w asemblerze.
Sekcje dzielimy wg. zawartości na;

- Sekcja inicjalizowana – (kod programu, predefiniowane stałe), czyli fragmenty programu, tablice o znanej na etapie programowania zawartości.
- Sekcja nieinicjalizowana – (rezerwacja obszarów pamięci na zmienne lub stałe wprowadzane w trakcie działania programu), czyli obszary, dla których na etapie pisania programu nie znamy zawartości a jedynie ich typ i zamieszczamy dane tylko o potrzebnej na nie objętości
- Sekcje mogą być opatrywane nazwami lub nie – mówimy wtedy o sekcja nazwanych lub nienazwanych

Sekcje są umieszczane przez linker we wskazanych obszarach pamięci zgodnie z zapisem zbioru konfiguracyjnego. Sekcje o tych samych nazwach łączone są we wspólne obszary ułatwiając organizację danych w pamięci.

2.79 Czym różnią się od siebie programy procedur obsługi przerwania ISR i dowolnej innej procedury?

Procedury obsługi przerwania – ISR –

- Mogą być wywoływane zdarzeniami lub sygnałami zewnętrznymi
- Działanie związane jest z priorytetami definiującymi ważność
- Mogą być wywoływane rozkazami TRAP n i INTR n
- Zawsze zaczynają się w tablicy wektorów przerwań
- Zawsze są wywoływane za pośrednictwem tablicy wektorów przerwań
- Mogą być wywołane w prawie dowolnym miejscu programu prawie niezależnie (!RPT!) od realizowanego rozkazu
- Ich wykonywanie zależy nie tylko od wystąpienia zdarzenia ale również od stanu indywidualnych masek oraz maski globalnej INTM (mogą zatem być blokowane)
- Zawsze blokują system przerwań maskowalnych poprzez ustawienie INTM
- Ich uruchomienie działania może zostać wstrzymane z powodu repetycji rozkazu
- Kończą się rozkazem powrotu z przerwania RETI

Standardowe procedury programowe

- Działanie ich zależy jedynie od przywołania i programu
- Nie są związane z priorytetami
- Są wywoływane bezpośrednio z programu rozkazami CALL
- Mogą się zaczynać w dowolnym miejscu przestrzeni pamięci programu
- Nie zależą od stanu maski INTM
- Nie są wstrzymywane działaniem repetycji
- Kończą się rozkazem powrotu RET

Obie odmiany procedur korzystają w podobny sposób ze stosu zapisując na nim adres powrotu po zakończeniu działania

2.80 Objaśnij składnię i zawartość linii poleceń asemblera, jakie pola wchodzą w jej skład i czemu służą?

W skład linii poleceń programu mogą wchodzić następujące elementy;

- Dyrektywa – polecenie dla asemblera, zwykle zaczynające się od kropki. Wpływa na proces asemblacji (tłumaczenia programu na język maszynowy).
- Etykieta – nazwa / tekst zaczynająca się w pierwszej kolumnie (od lewego marginesu). Zwykle odpowiada adresowi w pamięci lub wartości używanych potem w treści programu do odwołań i skoków. Oznacza- miejsca w programie, do których mogą następować skoki. Może reprezentować blok rozkazów, makro. Separatorem etykiety jest zwykle spacja lub średnik.
- Mnemonik – skrót mnemotechniczny o treści / brzmieniu nawiązującym do reprezentowanej operacji. Odpowiada rozkazowi i jest tłumaczony przez asembler na kod binarny procesora. Jest zachowana jednoznaczność pomiędzy każdym mnemonikiem a kodem rozkazu stąd możliwe jest jednoznaczne tłumaczenie w obu kierunkach (asemblacja i deasemblacja). Mnemonik kończy zawsze spacja lub średnik
- Operand (operandy) rozkazu – zapisane z użyciem odpowiedniej notacji dla użytego trybu adresacji wskazania na lokację lub wartość operandu rozkazu. Operandy oddzielane są od siebie przecinkami a kończą zwykle spacja lub średnik.
- Komentarz – zwykle rozpoczynający się średnikiem tekst objaśniający działanie pisanego fragmentu programu. Zawartość komentarza nie jest brana pod uwagę poprzez tłumaczące programy. Nie ma również swej reprezentacji w binarnym kodzie procesora. Komentarz kończy się zwykle z końcem linii (CR/LF)

[BP, str. 86]

2.81 W jaki sposób przekazuje się linkerowi polecenia i informacje jakie sekcje ma łączyć ze sobą, a jaki nie w C5515?

Polecenia dla linkera można przekazywać w różny sposób w zależności o ich rodzaju. Najprostsze można przekazać podczas jego wywołania w linii poleceń. Niektóre, np. dotyczące dołączania zbiorów

lub łączenia danych w bloki na jednej stronie, można zamieszczać w kodzie źródłowym programu. Najszersze możliwości i swobodę daje użycie zbioru konfiguracyjnego linkera (w CCS z rozszerzeniem .cmd), gdzie można praktycznie użyć wszystkich dostępnych poleceń.
Dołączaniem zbiorów można sterować w kodzie źródłowym z użyciem dyrektyw (np. .mmregs), a używając odpowiednio nazw sekcji wskazujemy linkerowi jakie z nich ma ze sobą scalać we wspólnych obszarach. W kodzie źródłowym można również posługiwać się flagami-kluczami np. by wymusić lokowanie bloku danych na tej samej stronie pamięci. (np. x .usect "vars3",4,**1** ;wytłuszczona „1" wymusza rezerwację miejsca dla 4 słów w pamięci na jednej stronie i przygotowuje w ten sposób do sprawnego dostępu do danych za pośredni adresacji bezpośredniej).
Znacznie szerszą gamę możliwości otwiera użycie zbioru konfiguracyjnego linkera. Można w nim zdefiniować zbiory łączone i zbiory generowane przez linker,
W zbiorze tym zamieszczamy również zapis podziału mapy pamięci na podobszary by następnie kierować w nie sekcje wg. typu lub nazw sterując w ten sposób ich łączeniem.

2.82 Do czego jest przeznaczona adresacja z odwróceniem bitów i w jakim trybie adresacji występuje?

Adresacja z odwróceniem bitów przeznaczona jest do usprawnienia obsługi adresacji w buforach próbek i współczynników w obliczeniach FFT. Realizuje mechanizm „motylków" FFT i jest zaimplementowana w trybie adresacji pośredniej.
Dalsze informacje można znaleźć w opisie pytania 2,24

2.83 Wymień podstawowe cykle występujące w działaniu procesora, czemu służą? Skąd wynika ich długość?

Wykonywanie każdego rozkazu procesora jest rozbite fazy. W przypadku procesora c5402 takich faz w wykonaniu każdego z rozkazów jest 6.
PREFETCH - Adres nowego rozkazu na linie adresowe magistrali programu
FETCH - Pobranie z pamięci rozkazu do analizy w procesorze
DECODE - Analiza treści rozkazu i ustalenie sposobu dalszego działania
ACCESS - Przygotowanie adresu operandów rozkazu
READ - Pobranie operandów i przygotowanie adresu odesłania wyniku
EXECUTE/ WRITE - Wykonanie rozkazu i odesłanie wyniku
Każda z tych faz jest realizowana w jednym cyklu procesora a na każdy cykl procesora przypadają 4 cykle zegara systemu.
Rozkazy są realizowane w kolejce i w trakcie wykonania zachodzą na siebie, ich czasy wykonania częściowo nakładają się na siebie (przetwarzanie nakładkowe). Dzięki takiemu sposobowi pracy wykonanie pojedynczego rozkazu może być rozliczone w programie jednym cyklem procesora. Warunkiem takiego rozliczenia jest pełne wykorzystanie kolejki.
[BP, str. 95], [C54x_CPU_Priph_spru131g.pdf; str. 7.1]

2.84 Co to jest trzeci stan dla wyjścia układu, czym się charakteryzuje i do czego służy?

Trzeci stan układu „Z" to tzw. stan wysokiej impedancji. Odróżnia się od podstawowych stanów „wysokiego" („1") i „niskiego" („0") tym, że nie wpływa na poziom linii sygnału, z którą jest połączony. W ten sposób można w przypadku wielu wyjść dołączonych do tej samej linii wybrać jedno, które będzie sterowało jej poziomem odłączając jednocześnie wszystkie inne wyjścia wprowadzone w stan „Z" dla uniknięcia konfliktu i uszkodzenia. Wymaganie od wyjścia możliwości wprowadzenia w trzeci stan jest podstawowym warunkiem dołączania wyjść do magistral.

2.85 O czym mówimy używając określenia „magistrala danych" czy „magistrala programu"?

Magistrala jest zespołem linii ze zdefiniowanym sposobem przesyłania po nich obiektów, dla których jest przygotowana. Charakterystyczną cechą magistrali jest podłączenie do niej wielu wejść i wyjść

układów równocześnie. W przypadku magistrali danych przewiduje się po niej transmisję danych, operandów instrukcji, stałych oraz wyników operacji pobieranych lub zapisywanych w pamięci danych. W przypadku magistrali programu przewidujemy przesyłanie po niej kodów programu – rozkazów i ich operandów z pamięci programu do rejestrów procesora. Dla prawidłowego działania tych magistral konieczne jest, by w ich skład wchodziły poza liniami transferu danych czy rozkazów linie adresowe – wybierające aktywne wyjście do sterowania jego stanami – i linie sterujące synchronizujące czas operacji transferu.
Do linii magistral mogą być podłączane jedynie wyjścia układów mogące poza dwoma podstawowymi stanami logicznymi „1" (wysoki, prawda) czy „0" (niski, fałsz) mogą przyjmować stan trzeci, „wysokiej impedancji" (dużej rezystancji pomiędzy wyjściem a masą i zasilaniem) odłączający to wyjście od sterowania stanem linii (pozwalający na to sterowanie innemu wyjściu również podłączonemu do tej samej linii).
[BP, str. 25]

2.86 Kilka uwag na koniec

W przedstawionym powyżej materiale można spotkać w niektórych opisach odesłania do literatury. Te uzupełnienia ze względów czasowych są jednak nie kompletne. Zastosowałem trochę uproszczoną konwencję odsyłaczy umieszczając w kwadratowych nawiasach skrót tytułu źródła i stronę, gdzie można znaleźć informacje. I tak;

[V3.x TechRef] dokumentacja producenta – TI – ***C55x v3.x Technical Overview*** - na stronie kursu widoczne jako `C55xx_CPU_RG_spru371f.pdf`

[V3.x CPU] dokumentacja producenta –TI – ***C55x v3.x CPU-Reference Guide*** - na stronie kursu widoczne jako `C55xx_v3.x CPU RefGuide_swpu073e.pdf`

[V3.x InstSet_RG] dokumentacja producenta –TI – „***C55x v3.x CPU Mnemonic Instruction Set Reference Guide***" na stronie kursu widoczna jako `C55xx_MnemoInstrSet_v3_RG_swpu067e.pdf`

[C5515_Data] dokumentacja producenta –TI – „***TMS320C5515 Fixed-Point Digital Signal Processor (Jan. 2012)***" na stronie kursu widoczna jako `TMS320C5515_DOC_sprs645e.pdf`

[C55xx_SysUG] dokumentacja producenta –TI – „***TMS320C5515 DSP System Users Guide (Rev. D)***" Dokumentacja TI dostępna na stronie kursu widoczna jako `TMS320c5515DSP_UG_sprufx5d.pdf`

[C55xx_Mnem_InSRG] dokumentacja producenta –TI – „***TMS320C55xx DSP Mnemonic Instruction Set Reference Guide***" na stronie kursu widoczna jako `C5515_Mnemonic_InS_RG_spru374g.pdf`

[C55xx_AssLangTools] dokumentacja producenta –TI – „***TMS320C55x Assembly Language Tools v 4.4 Users Guide***" na stronie kursu widoczna jako `C55xx_AssLangTools_v4.4_UG_spru280i.pdf`

[C55xx_Periph] dokumentacja producenta –TI – „***TMS320C55x DSP Peripherals Overview"*** – skierowania do dokumentacji peryferii rodziny procesorów C55xx, na stronie kursu widoczna jako `C55xx_Periph_OV_UG_spru317k.pdf`

Dokumentacje te można znaleźć na stronie kursu w katalogu „Zbiory z dokumentacją procesora wykładowego++"

Udostępniany materiał jest złożeniem i przystosowaniem opracowań kilku roczników. Jest praktycznie zbiorem pytań i zagadnień, które występowały w treści dotychczasowych cykli zaliczeń i egzaminów. Układ i podział na działy pochodzi od autora początkowego materiału. Uznałem, że wyznaczony został trudnościami i potrzebami interpretacji. Stąd zachowałem go.

Warto jeszcze dodać, że materiał kursu sięga nie tylko do w/w dokumentacji ale i do rozdziałów trzech książek;

[BP, str. …] „**Intruduction to Digital Signal Processors**” Bruno Paillard

[RTDSP-Kuo-Lee] „**Real-Time Digital Signal Processing**” S.M.Kuo, B.H.Lee (rozdział 2),

[DSPFaA-Tan] „**Digital Signal Processing – Fundamentals and Applications**” Li Tan (rozdział 9).

Książka i oba rozdziały są dostępne na stronie kursu i obejmują następujące obszary;

- Budowa i własności toru DSP
- Budowa procesora (C5515) i jej rozumienie (Architektura – budowa procesorów, ich główne bloki charakteryzujące działanie i pozwalające na odróżnienie odmian, zadania głównych bloków i ograniczenia, przeznaczenie, elementy schematu blokowego, rejestry ich przeznaczenie, magistrale, sekwencje operacji na magistralach, przestrzenie pamięci, portów, rodzaje pamięci, mapa pamięci, …)
- Działanie i mechanizmy (mechanizmy wykorzystywane w działaniu, usprawniające działania, pozwalające na sprawniejsza obsługę strumienia danych, DMA, stos, przerwania, bufory Ping-Pong, warunki sprawdzane w procesorze, sposoby ich wykorzystania).
- Obliczenia, rozumienie głównych rozkazów, (cykle, realizacja rozkazu, kolejki, repetycja, mechanizmy adresacji, realizacje pętli, w tym również działanie podstawowych rozkazów, dalej reprezentacja danych i operacje na nich, konwersje, arytmetyka obliczeń, mechanizmy korekcyjne, …)
- Podstawowe operacje DSP (MAC i jej realizacja, filtracja, transformacje, FFT, …)
- Konfigurację Narzędzi środowiska developerskiego (programy narzędziowe, emulatory ICE, moduły developerskie, DSK, EVM, biblioteki, ich przeznaczenie i możliwości, CSL, BSL, DSPLib, IQmath, …)

Dlatego sugeruję raczej przygotowywanie do zaliczenia „przez problemy i zagadnienia” z przeglądem uwag na temat w książce i dokumentacji. Pozostanie przy jednej pozycji literatury a tym bardziej tylko przy foliach ilustracji wykładu jest mało efektywne.

Spis pytań.