

1. Tabelki

Poniższe rozwiązania obu tabelki urodziły się na podstawie opracowań zrobionych na forum (podziękowania w stronę Kornika, Pachooka, Freddiego, Bonanzy, Koompla i wszystkich, którzy dołożyli coś od siebie w tej kwestii), oraz zostały skorygowane o poprawki i tłumaczenia dr Kardacha uzyskane podczas oględzin pracy po pierwszym terminie, zatem teraz są zrobione w 100% dobrze.

Tabela „arytmetyczna”

w.		DEC	HEX	BIN	12-bitowy akumulator
	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

Rozwiązanie dotyczy przykładowej tabelki z grupy A z 2004 roku. Dotyczą jej pierwsze 4 zadania na kolokwium:

1. Dla liczb **A=0,75** i **B=-0,125** uzupełnić tabele w kolumnach 2,3,4 zakładając notację **U2** i format **I1Q5**.

2. Zakładając pracę procesora na słowie **6-bitowym** i kodowanie **U2, I1Q5**, oraz akumulator **12-bitowy** i włączony **SXM** uzupełnij tabelę w kolumnie 5 wykonując odpowiednie rozkazy:

dla wiersza 1 **LD#WA,2,A**

dla wiersza 2 **LD#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. Kodujemy **A** i **B** w kodzie **U2** (jeśli ktoś do tej pory nie wie, jak się to robi, to niech zajrzy tutaj <http://www.i-lo.tarnow.pl/edu/inf/alg/num/pages/018.php>). Inne przydatne materiały odnośnie systemów liczbowych i ich kodowania można znaleźć na poniższych stronach:

<http://network.page.com.pl/materiały/Konwersje.htm>

<http://www.i-lo.tarnow.pl/edu/inf/alg/num/pages/014.php>

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. Tak więc:

$$WA = 0,75_{10} = 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5} = 011000_2$$

$$\mathbf{WB} = -0,125_{10}$$

Aby uzyskać **B**, kodujemy **0,125** jak wyżej, następnie negujemy wszystkie bity i dodajemy **1** do pozycji najmłodszego bitu:

$$0,125_{10} = 0 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5} = 000100_2$$

$$\sim 0,125_{10} = 111011_2$$

$$\begin{array}{r} 111011 \\ + \quad 1 \\ \hline 111100 \end{array}$$

zatem $\mathbf{WB} = -0,125_{10} = 111100_2$. Teraz można wykonać następujące działania:

▪ **dodawanie**

$$\begin{array}{r} 011000 \\ + 111100 \\ \hline 1010100 \end{array}$$

Nawet jeżeli w wyniku jakiegoś działania wyjdzie nam przeniesienie na kolejny, nieistniejący bit (tu zaznaczony na żółto), to nie uwzględniamy tego w wyniku, czyli prawidłowo jest:

$$\mathbf{WA + WB = 010100_2 = 0,625}$$

Analogicznie jest dla odejmowania (uwzględniamy tylko 6 najmłodszych bitów wyniku) i mnożenia (uwzględniamy tylko 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:

$$\begin{array}{r} 011000 \\ + 000011 \\ + \quad 1 \\ \hline 011100 \end{array}$$

$$\mathbf{WA - WB = 011100_2 = 0,875}$$

UWAGA: jeżeli w wyniku odejmowania dziesiętnego otrzymana zostanie liczba mniejsza od **-1**, w wyniku odejmowania binarnego otrzymamy różną od niej liczbę! Dzieje się tak z powodu przekroczenia zakresu dozwolonych wartości dla formatu **I1Q5** (pozwala on na minimalną wartość równą **-1**). **To jest celowy haczyk i należy napisać o tym w tabelce lub pod nią.**

- **mnożenie** - na potrzeby mnożenia w **U2** należy dwukrotnie zwiększyć ilość bitów liczb **A** oraz **B** poprzez dostawienie **6 starszych bitów** wypełnionych wartością taką, jaką ma **znak** danej liczby w **U2**:

$$\mathbf{WA = 011000_2 \rightarrow 000000 \ 011000_2}$$

$$\mathbf{WB = 111100_2 \rightarrow 111111 \ 111100_2}$$

Po dokonaniu tego można pomnożyć **WA** • **WB** (tu oczywiście wygodniej będzie wymnożyć **WB** • **WA**):

```

111111111100
·000000011000
-----
000000000000
000000000000
000000000000
1111111100
111111100
...

```

111110100000

WA • WB = 111101₂ = -0,09375

Wynikiem mnożenia, który należy zamieścić w tabelce w wierszu 5, kolumnie 4 są bity na pozycjach zaznaczonych powyżej na czerwono.

Mając już wyniki dziesiętnie i binarnie, możemy łatwo przekonwertować BIN na HEX. Z racji tego, że są tu 2 różne drogi rozwiązania (najprawdopodobniej obie prawidłowe pomimo tego, że dają różne wyniki), przedstawię tę metodę, której jestem w 100% pewien, gdyż zastosowałem ją na kolokwium i została oceniona jako poprawna.

Nad kolumną HEX w tabelce robimy adnotację np. typu „HEX w I1Q5, dopełnienie zerami do 8 bitów”, a następnie przekształcamy BIN na HEX po uprzednim do-
stawieniu **2 najstarszych bitów** będących zerami, **niezależnie od tego czy prze-
kształcana liczba jest dodatnia czy ujemna** (przykład dla **WA**):

WA = 011000₂ = 0001 1000₂

1 8

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

011000₂ = 18₁₆

ad.2. Bit SXM wpływa na to, jak uzupełniane są najstarsze bity w akumulatorze:

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

Mamy wykonać następujące rozkazy:

LD#WA,2,A ← załaduj **WA** do akumulatora i przesun o **2** w lewo

LD#WB,A ← załaduj **WB** do akumulatora

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

- ładujemy **WA** do akumulatora: _ _ _ _ _ **011000**
- przesuwamy o 2 pozycje w lewo: _ _ _ _ **011000** _ _
- uzupełniamy **najstarsze bity** w zależności od **znaku** (czyli tutaj zerami), a na **najmłodszych** **zawsze** wstawiamy zera: **000001100000**

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

111111111100

ad.3. Dla dodawania i odejmowania postępujemy tak, jak w punkcie poprzednim, pamiętając o włączonym bicie SXM, czyli po prostu ładujemy wyniki do akumulatora i odpowiednio uzupełniamy zerami lub jedynekami w zależności od znaku. W przypadku mnożenia po prostu przepisujemy do akumulatora to, co otrzymaliśmy w wyniku pisemnego mnożenia binarnego, czyli w naszym wypadku:

111110100000

ad.4. Bit FRCT powoduje skasowanie „nadmiarowego” znaku liczby, co dzieje się poprzez przesunięcie liczby o **1** w lewo, czyli jeśli mamy w akumulatorze wynik mnożenia:

111110100000

to po przesunięciu o jeden w lewo mamy:

111101000000

Powstałą wskutek przesunięcia „pustą” pozycję na najmłodszym bicie **zawsze** uzupełniamy **zerem**.

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

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

w.		DEC	HEX	BIN	12-bitowy akumulator
	1	2	3	4	5
1	WA	0,75	0x18	011000B	000001100000
2	WB	-0,125	0x3C	111100B	111111111100
3	WC=WA+WB	0,625	0x14	010100B	000000010100
4	WC=WA-WB	0,875	0x1C	011100B	000000011100
5	WC=WA*WB	-0,09375	0x3D	111101B	111110100000

Poniżej pozostałe 3 tabelki z grup **B, C, D** z 2004 roku.

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

w.		DEC	HEX	BIN	12-bitowy akumulator
	1	2	3	4	5
1	WA	-0,875	0x24	100100B	111110010000
2	WB	0,5	0x10	010000B	000000010000
3	WC=WA+WB	-0,375	0x34	110100B	111111101010
4	WC=WA-WB	-1,375*	0x14	010100B	000000010100
5	WC=WA*WB	-0,4375	0x32	110010B	111001000000

*przekroczenie zakresu; dziesiętnie wychodzi -1,375, zaś binarnie 0,625

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

w.		DEC	HEX	BIN	12-bitowy akumulator
	1	2	3	4	5
1	WA	0,125	0x04	000100B	000000010000
2	WB	-0,75	0x28	101000B	111111101000
3	WC=WA+WB	-0,625	0x2C	101100B	111111101100
4	WC=WA-WB	0,875	0x1C	011100B	000000011100
5	WC=WA*WB	-0,09375	0x3D	111101B	111110100000

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

w.		DEC	HEX	BIN	12-bitowy akumulator
	1	2	3	4	5
1	WA	-0,25	0x38	111000B	111111100000
2	WB	0,375	0x0C	001100B	000000001100
3	WC=WA+WB	0,125	0x04	000100B	000000000100
4	WC=WA-WB	-0,625	0x2C	101100B	111111101100
5	WC=WA*WB	-0,09375	0x3D	111101B	111110100000

Tabela „rozkazowa”

Sprowadza się do odpowiedniej interpretacji rozkazów - pod tabelką instrukcja krok po kroku. **Wszystkie liczby wstawione do tabelki podczas rozwiązywania są w HEX, a nie w DEC!**

Dane:	DP=0		DP=2		DP=4	
CPL=0	60	60	200	120	300	130
CMPT=0	61	40	201	70	301	80
Adr./Dane HEX	62		202	20	302	100
	Adres	Wartość	Adres	Wartość	Adres	Wartość

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

Postępowanie dla kolejnych wierszy tabeli:

1. Załaduj liczbę binarną 0 (co daje 0h) do DP.
2. Załaduj liczbę binarną 2 (co daje 2h) do AR0.
3. Załaduj liczbę 200h do AR1.
4. Załaduj liczbę 300h do AR2.
5. Załaduj wartość komórki o adresie 61 do A (patrz nad tabelką).
6. Załaduj do pamięci zawartość komórki o adresie równym wartości AR1 (zawartość AR1 to 200, a odpowiadająca temu adresowi wartość komórki to 120 – patrz nad tabelką), dodaj do tego wartość A, zapisz wynik w A i inkrementuj zawartość AR1.
7. Załaduj wartość komórki o adresie 60 (adres 60 zawartość równa 60 – patrz nad tabelką) do pamięci, odejmij to od A i wynik zapisz w B.

8. Załaduj do pamięci zawartość komórki o adresie równym wartości AR1 (zawartość AR1 to 201, a odpowiadająca temu adresowi wartość komórki to 70 – patrz nad tabelką), dodaj to do wartości B i zapisz wynik w A, inkrementuj zawartość AR1.
9. Załaduj liczbę 4 do DP.
10. Mając DP=4 (zrobiliśmy to w wierszu 9) patrzymy do tabelki nad główną tabelką i szukamy DP=4 (pierwsza mała tabelka od prawej strony). @2 w wierszu 10 mówi, że interesuje nas adres kończony się na 2, a więc w tym przypadku adres 302 (wybieramy dla DP=4 adres kończący się na 2). Do pamięci ładujemy wartość komórki o adresie 302. Wartość komórki o adresie 302 to 100. Dodajemy to do A i wynik zapisujemy w A.
11. Załaduj do pamięci zawartość komórki o adresie równym wartości AR2 (zawartość AR2 to 300, a odpowiadająca temu adresowi wartość komórki to 130 – patrz nad tabelką) dodaj to do wartości A i zapisz wynik w A, inkrementuj zawartość AR2.
12. Załaduj do pamięci zawartość komórki o adresie równym wartości AR2 (zawartość AR2 to 301, a odpowiadająca temu adresowi wartość komórki to 80 – patrz nad tabelką), odejmij od wartości A i zapisz wynik w A, inkrementuj zawartość AR2.
13. Załaduj do pamięci binarna wartość 64 (co daje 40h). Odejmij to od A i zapisz wynik w A.
14. Załaduj do pamięci zawartość komórki o adresie równym wartości AR2 (zawartość AR2 to 302, a odpowiadająca temu adresowi wartość komórki to 100 – patrz nad tabelką). Dodaj to do A i wynik zapisz w B. Teraz *AR2-0 oznacza: od tego, co jest w AR2, odejmij to, co jest w AR0 (302h-2h=300h) i zapisz w AR2.
15. Załaduj do pamięci zawartość komórki o adresie równym wartości AR2 (zawartość AR2 to 300, a odpowiadająca temu adresowi wartość komórki to 130 – patrz nad tabelką). Odejmij to od B i zapisz wynik w A.
16. Załaduj liczbę binarną 160 (co daje A0h) do AR0.
17. Załaduj do pamięci zawartość komórki o adresie równym wartości AR1 (zawartość AR1 to 202, a odpowiadająca temu adresowi wartość komórki to 20 – 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 AR1.

UWAGA: w powyższej tabelce znajduje się kolejny haczyk. Nad tabelką puste miejsce na wstawienie wartości jest przy adresie 62, a z 18 wiersza tabelki wynika, że trzeba załadować wartość znajdującą się w akumulatorze do komórki o adresie 162. Jeżeli się tak zdarzy, należy obok tabelki narysować poglądowo fragment pamięci z komórką o adresie 162, do której ma trafić wartość zawarta w akumulatorze, np. tak jak poniżej:

.	
.	
160	
161	
162	2D0
.	
.	

Poniżej wypełnione tabelki dla grup **C** i **D** z 2004 roku.

Dane:	DP=0		DP=2		DP=4	
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
	Adres	Wartość	Adres	Wartość	Adres	Wartość

Program	A	B	DP	AR0	AR1	AR2
LD #0,DP			0			
STM #2,AR0				2		
STM #100h,AR1					100	
STM #200h,AR2						200
LD @0x61,A	40					
ADD *AR1+,A	160				101	
SUB @60h,A,B		100				
ADD *AR1+,B,A	160				102	
LD #4,DP			4			
ADD @1,A	1B0					
ADD *AR2+,A	2E0					201
SUB *AR2+,A	290					202
SUB #64,A	250					
ADD *AR2-0,A,B		350				200
SUB *AR2,B,A	220					
STM #160,AR0				A0		
ADD *AR1-0,A,A	240				62	
STL A,*AR1-					61	

Dane:	DP=0		DP=2		DP=4	
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
	Adres	Wartość	Adres	Wartość	Adres	Wartość

Program	A	B	DP	AR0	AR1	AR2
LD #0,DP			0			
STM #2,AR0				2		
STM #100h,AR1					100	
STM #200h,AR2						200
LD @0x61,A	140					
ADD *AR1+,A	260				101	
SUB @60h,A,B		200				
ADD *AR1+,B,A	230				102	
LD #4,DP			4			
ADD @1,A	2A0					
ADD *AR2+,A	3D0					201
SUB *AR2+,A	360					202
SUB #64,A	320					
ADD *AR2-0,A,B		420				200
SUB *AR2,B,A	2F0					
STM #160,AR0				A0		
ADD *AR1-0,A,A	310				62	
STL A,*AR1-					61	

2. Pytania z testów - opracowanie

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

- ultra małe obudowy (BGA, TQFP)
- moc obliczeniowa pojedynczego rdzenia DSP rzędu 160 MIPS
- rozmiar pamięci do 8 megabajtów
- sprzętowe nasycanie i zaokrąglanie
- sprzętowa jednostka mnożąca (MAC)
- zwielokrotnienie i specjalizacja magistral
- osobna magistrala danych i programu procesora
- specjalizowane rozkazy do przetwarzania sygnałów (FIRS, SUBC, POLY...)

Jaka jest w procesorach C54xx rola rejestrów statusowych i dlaczego jest ich aż trzy?

Rejestry te służą do zachowania informacji o stanie pracy procesora i wybranych ustawieniach. Jest ich aż trzy (ST0, ST1, PMST) z racji tego, że zachodzi potrzeba przechowania liczby informacji, która nie da się przechować w mniejszej liczbie rejestrów.

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

Na przykładzie różnic między C54xx a C55xx:

- zwielokrotnienie MAC i ALU
- zwielokrotnienie akumulatora
- dodatkowy, trzeci generator adresów
- dodatkowe dwie magistrale adresowe

Można również stwierdzić, że ma na to wpływ:

- rozdzielenie pamięci danych i programu
- osobne magistrale programu i danych
- zwielokrotnienie magistrali do odczytu
- osobna magistrala do zapisu

Od czego można uzależnić przebieg programu w procesorach rodziny C54xx?

Od następujących rozkazów sterujących przebiegiem programu:

- instrukcji skoku (B, BACC, BANZ, BC)
- instrukcji wywołań procedur (CALL, CALA, CC)
- instrukcji odpowiedzialnych za obsługę przerwań (INTR, TRAP)
- instrukcji powrotu (RETE, RET)
- instrukcji manipulujących danymi na stosie (FRAME, POPD, PSHD, PSHM...)
- instrukcji odpowiedzialnych za repetycję (RPT, RPTB, RPTZ)
- innego typu instrukcji (RESET, SSBX, RSBX, NOP, IDLE)

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. Dodatkowe bity ponad ten dwukrotny rozmiar wykorzystywane są na guard dla obsługi wyników pośrednich.

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

Przetwarzanie nakładkowe, czyli pipelining, pozwala na przyspieszenie wykonywania instrukcji, ponieważ umożliwia procesorowi jednoczesne równoległe wykonywanie kilku faz cyklu rozkazowego. Przykładowo, wykonując fazę Pre-Fetch dla jednej instrukcji, procesor może jednocześnie wykonywać fazę Fetch poprzedniej instrukcji, fazę Decode dla jeszcze wcześniejszej instrukcji itd. Średnio procesor przetwarza równocześnie 6 instrukcji, dodając nową co kolejny cykl zegara. Wielofazowe nakładkowanie poprawia wykorzystanie DSP.

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 powodować konflikt zewnętrznego odczytu danej z pobraniem rozkazu w pipeliningu. Może to spowodować obniżenie efektywności o co najmniej 50%.

Dlaczego w procesorze DSP stosuje się wiele równoległych magistral transportowych?

Dlatego, że równoległy odczyt danych (2 magistrale do odczytu) i zapis przyspieszają wykonanie (w 1 cyklu można zrobić odczyt 2 danych i zapis trzeciej). Przekłada się to na szybszą realizację rozkazów.

Co to jest DARAM i dlaczego jest ona korzystna w procesorach DSP C54xx?

DARAM (Double Access RAM) jest to pamięć zezwalająca na 2 dostępy na blok na cykl, co oznacza, że zarówno CPU jak i peryferia mogą dokonywać odczytu i zapisu w tym samym cyklu. Istotne jest ponadto, że pamięć DARAM można przełączyć z wykorzystaniem bitu OVLY do przestrzeni pamięci programu.

Wymień tryby adresacji stosowane w rodzinie procesorów TMS320C54xx i podaj przykłady rozkazów stosujących je.

Adresacja	Przykład	Przeznaczenie, zalety
Natychmiastowa (Immediate)	LD #10,A	- operand bezpośrednio w kodzie rozkazu - użyteczne do inicjalizacji
Absolutna (Absolute)	STL A,*(y)	- używa 16-bitowego adresu dowolnej komórki - wymusza dwusłowy rozkaz
Akumulatorem (Accumulator)	READA x	- adresem operandu w pamięci programu jest zawartość akumulatora
Pośrednia (Indirect)	LD *AR1,A	- adresem operandu jest zawartość aktywnego rejestru (ARi) użyta jako wskaźnik
Bezpośrednia (Direct)	LD @x, A	- adresacja względem wskaźnika strony -DP albo wskaźnika stosu -SP
Stosowa (Stack)	PSHM AG	- push / pop danej z pamięci danych lub z MMRs (rejestrów widocznych w przestrzeni pamięci)
MMR	LDM ST1,B	- adresacja z użyciem nazwy rejestru MMR - szybki dostęp do rejestrów MMR

Jak rozumiesz i co określa pojęcie trybu adresacji?

Tryb adresacji jest sposobem podawania adresu w rozkazie. Określa on, w jaki sposób instrukcje dostają się do swoich operandów w pamięci. Wyróżniamy następujące tryby adresacji:

- natychmiastowy LD #10,A
- absolutny LD A, *(y)
- akumulatorem READ A x
- pośredni LD *AR1,A
- bezpośredni LD @x,A
- za pomocą stosu
- MMR LDM ST!,B

Wymień sposoby modyfikacji zawartości rejestrów adresowych procesorów C54xx i podaj ich przykładowe przeznaczenie.

Opcja	Składnia	Realizacja
Bez modyfikacji	*ARn	Arn bez zmian
Inkrement / Dekrement	*ARn+ *ARn-	post inkrement o 1 post dekrement o 1
Indeksowana	*ARn+0 *ARn-0	post inkrement o zawartość AR0 post dekrement o zawartość AR0
Kołowa (Circular)	*ARn+% *ARn-% *ARn+0% *ARn-0%	post inkrement o 1 - kołowo post dekrement o 1 - kołowo post inkrement o AR0 - kołowo post dekrement o AR0 - kołowo
Z odwr.bitów (Bit-Reversed)	*ARn+0B *ARn-0B	post ink. ARn by AR0 z odwr.bitów post dek. ARn by AR0 z odwr. bitów
Pre-modyfik.	*ARn (lk) *+ARn (lk) *+ARn (lk)% *+ARn	*(ARn+LK), ARn bez zmian! *(ARn+LK), ARn zmienione *(ARn+LK), ARn zmienione – kołowo pre-inkrement o 1, tylko dla zapisu
Absolutna	*(lk)	16-bit stała lk użyta jako adres absolutny Patrz adresacja absolutna

Przykładowe zastosowania:

- inkrement/dekrement - tablice, sygnały liniowe
- kołowe - przy wpisywaniu danych do jednego obszaru
- z odwróceniem bitów - do szybkiej transformaty Fouriera

Co to są sekcje programu i do czego są używane?

Sekcje to moduły zawierające jednorodne obiekty; kod, stałe, zmienne lub układy we/wy. Są one zdefiniowane za pomocą dyrektyw w zbiorach źródłowych. Sekcje są umieszczone we wskazanych obszarach pamięci przez linker. Wyróżniamy następujące rodzaje sekcji:

- sekcja inicjalizowana
- sekcja nieinicjalizowana
- sekcja nazwana
- sekcja nienazwana

Co to jest dyrektywa programu i do czego służy?

Dyrektywa programu jest to polecenie dla assemblera, nakazujące mu określone działanie i rozpoczynające się od kropki. Dyrektywa jest elementem sterowania assemblera, który nie pozostawia śladu w kodach wynikowych programu (po asemblacji). Dyrektywy mogą służyć np. do zdefiniowania sekcji w zbiorach źródłowych. Przykładem dyrektywy może być:

`.sect „nazwa”` tworzy inicjalizowaną i nazwaną sekcję na kod programu i dane

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

Linker łączy plik *.asm i generuje plik wyjściowy *.obj. Rozmieszcza on sekcje we wskazanych obszarach pamięci i może generować *.map – mapę pamięci. Zajmuje się on relokacją symboli i sekcji, by przypisać je do ostatecznych adresów oraz decyduje o zewnętrznych powiązaniach między plikami wejściowymi. Do prawidłowego działania linkera niezbędny jest zbiór konfiguracyjny linkera - Linker Command File.

Wymień czynniki decydujące o szybkości realizacji programu w DSP.

- pipeline
- rozkazy specjalizowane i ukierunkowane na aplikacje
- rozmieszczenie danych w pamięci (zewnętrznej lub wewnętrznej)
- zwielokrotnienie magistral
- ilość przerwań, które są używane
- operacje dwusłowe
- zastosowanie adresacji kołowej lub z odwracaniem bitów
- opóźnione skoki, łączone warunki operacji
- zaawansowana obsługa pośrednich wyników operacji

Omów sposoby realizacji pętli i stosowane tam rozkazy.

Do realizacji pętli mogą zostać wykorzystane następujące metody:

- repetycja pojedynczego rozkazu realizowana instrukcją RPT lub RPTZ, pozwala na powtórzenie instrukcji od 1 do 65536 razy. Różnica między RPT a RPTZ polega na tym, że instrukcja RPTZ resetuje akumulator A lub B przez rozpoczęciem pętli.
RPT n → n+1 powtórzeń
- repetycja bloku rozkazów realizowana za pomocą instrukcji RPTB. Pozwala ona na powtórzenie bloku instrukcji od 1 do 65536 razy. Numer iteracji dany jest przez zawartość BRC (Block Repeat Counter) plus jeden.
BRC=n → n+1 powtórzeń
- instrukcje skoku warunkowego, wykonujące skok tylko wtedy, gdy spełniony jest dany prosty lub złożony warunek (w przeciwnym razie wykonanie programu przechodzi do następnej instrukcji). Wyróżniamy dwie instrukcje skoku warunkowego:

- ## Co to są tryby repetycji i czemu służą w procesorach DSP rodziny C'54xx?

Repetycja pojedynczego rozkazu:

- RPT $n \rightarrow n+1$ powtórzeń

- realizacją za pomocą instrukcji RPTB. Pozwala ona na powtórzenie bloku instrukcji od 1 do 65536 razy. Numer iteracji dany jest przez zawartość BRC (Block Repeat Counter) plus jeden.

BRC=n \rightarrow n+1 powtórzeń

a) ile cykli upłynie do wykonywania pierwszej instrukcji z wywołaniem procedury - 4 cykle (w przypadku CALLD też będą 4 cykle)

b) ile cykli potrzeba na realizację całej sekwencji - 9 cykli

c) ile słów zajmują te rozkazy w pamięci programu - 4 (2 słowa CALL + 2 słowa STM), w przypadku z CALLD będzie tak samo

d) ile słów zajmują te rozkazy w pamięci danych - 0 słów (tak samo CALLD)

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

12

czątkowego zawsze wynosi zero, co powoduje, że tablica przerwań zawsze rozpoczyna się pod adresem będącym wielokrotnością 128.

Mechanizm taki jest zaimplementowany z tego powodu, że zazwyczaj kod użytkownika implementuje własne przerwania, i nie może w związku z tym używać domyślnych wektorów przerwań rezydujących w ROM na chipie. Umożliwienie przesunięcia tablicy wektorów do dowolnej 128-słowej przestrzeni w pamięci programu rozwiązuje ten problem.

Co to jest i czemu służy w procesorach rodziny C'54xx IPTR?

Rejestr IPTR (czyli interrupt pointer) służy do wskazywania na początek tablicy wektorów przerwań. Starsza część adresu tablicy wektorów przerwań (najstarsze 9 bitów) tworzy adres początkowy tablicy przerwań procesora (adres położenia pierwszego rozkazu do wykonania) równy FF80h – bo jego zawartość umieszcza na jest na najstarszych bitach uprzednio wyzerowanego rejestru PC). Programista może przesunąć położenie tablicy wektorów przerwań w pamięci z wykorzystaniem IPTR.

Co to jest tablica wektorów przerwań i do czego ona służy?

Tablica wektorów przerwań znajduje się w pamięci programu (domyślnie rozpoczyna się pod adresem FF80h). Znajdują się w niej adresy podprogramów obsługi przerwań. Służy ona do wyboru odpowiedniego podprogramu przy uruchomieniu przerwania. Każdy wektor ma zawsze 4 słowa.

Co to jest przerwanie?

Przerwanie (ang. interrupt) – sygnał powodujący zmianę przepływu sterowania, niezależnie od aktualnie wykonywanego programu. Sygnalizowanym zdarzeniem może być np. przepełnienie licznika lub koniec transmisji danych. Przerwanie może wystąpić w dowolnym momencie niezależnie od programu. Pojawienie się przerwania powoduje wstrzymanie aktualnie wykonywanego programu i wykonanie przez procesor kodu procedury obsługi przerwania (ang. interrupt handler).

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

By zareagować na przerwanie trzeba ustawić INTM, IMR, SP i odpowiedni wektor. Procedura obsługi przerwania jest programem uruchamianym po wywołaniu przerwania (ustawieniu flagi przy ustawionej masce). Jej cechy to:

- stały adres początkowy w tablicy wektorów przerwań
- początkowa długość w tablicy wektorów ma rozmiar 4 słów (tam trzeba wykonać skok, by w ogóle rozpocząć obsługę przerwania)
- konieczne jest na wstępie zachowanie rejestrów na stosie i pobranie ich na końcu w kolejności odwrotnej do ich składania
- automatycznie wyłączany jest INTM. zatem inne przerwania nie mogą wystąpić
- automatycznie na stos odsyłany jest tylko PC
- ustawiany jest na zewnątrz stan aktywny potwierdzający rozpoczęcie obsługi przerwania

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

Wiąże je to, że obie muszą być ustawione, aby nastąpić procedura obsługi przerwania.

Różnica polega na tym, że flaga jest ustawiana sprzętowo, zaś maska ustawiana programowo; dodatkowo, są one w innych rejestrach.

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

Operacje te dotyczą zachowania i odtworzenia stanu części rejestrów procesora w trakcie ISR. Dotyczy to rejestrów ST0, ST1, oraz jeśli uruchomiona możliwość innego przerywania - również IMR. Operacje te podlegają one następującym zasadom:

rozkaz	opis
PSHM mmr	SP -1 → SP odesłanie MMR na STOS
POPM mmr	pobranie ze STOSu do MMR SP + 1 → SP
PSHD Smem	SP -1 → SP odesłanie daną z pamięci na STOS
POPD Smem	pobranie ze STOSu do pamięci danych SP + 1 → SP
FRAME K	modyfikacja wskaźnika stosu SP SP + K → SP

Należy zwrócić uwagę, że trzeba pamiętać o odwrotnej kolejności pobierania od kolejności zachowania na stosie.

Jaka jest zasada działania stosu i do czego on służy?

Stos charakteryzuje się odwrotną kolejnością pobierania danych ze stosu od kolejności ich zachowania

Definiowanie stosu:

1. Zadeklarować nieinicjalizowaną sekcję odpowiedniego rozmiaru.
2. Zainicjować wskaźnik stosu (SP) by wskazał " szczyt stosu +1":
3. Ulokować stos w pamięci (najlepiej w pamięci wewnętrznej)

SP wskazuje ostatnią zajętą komórkę, zatem dla;

CALL: PC → *--SP

RET: *SP++ → PC

Stos może być wykorzystany na przykład do zachowania rejestrów (context save/context restore) podczas ISR.

Jakie warunki można sprawdzać w procesorze C54xx , czego one dotyczą i jakie rozkazy mogą wykorzystywać ich wyniki.

W procesorze C54xx można sprawdzać następujące warunki:

- operacje logiczne AND, OR, XOR (przy czym ANDM, ORM, XORM działają bezpośrednio na pamięci danych)
- bity (BIT, BITT)
- pola kodów (BITF)
- warunki równości (CMPM, CMPR)

Do czego służy w procesorach DSP zegar (timer)?

Zegary w DSP obsługują:

- zdarzenia czasowe (np. pomiar czasu trwania funkcji czy innych procesów software'owych)

- generację impulsów i pomiar ich szerokości
- generację przerw
- zdarzenia synchronizujące dla DMA
- generację sygnałów okresowych do sterowania peryferiami typu ADC/DAC

Co odróżnia standardowy port szeregowy od McBSP w C54xx?

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.
- maksymalna szybkość bitowa: 1/2 CPU Clock Rate
- długość słowa: 8-, 12-, 16-, 20-, 24-, 32-bit
- praca wielokanałowa maksymalnie do 128 kanałów
- wbudowany komparing wg. praw μ lub A
- wewnętrzna generacja zegara/ramek z SGR (Sample Rate Generator)
- programowana polaryzacja zegara / ramek
- potrafi sygnalizować wszystkie typowe błędy i statusy

Na czym polega konfigurowanie do pracy peryferii w procesorach DSP?

Polega na zdefiniowaniu odpowiednich parametrów (głównie przy użyciu 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)

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 automatyczny transfer danych między portami McBSP a RAM zawartym na chipie, co znacznie upraszcza zarządzanie wielokanałowymi portami szeregowymi przez procesor.

Co to jest i do czego służy emulator procesora DSP?

Emulator procesora DSP umożliwia w zasadzie bardzo podobne funkcje jak debugger. Różnica polega na tym, że w przypadku debuggера 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. Wykorzystanie emulatora przekłada się na szybsze działanie oraz możliwość sprawdzenia programu w prawdziwych warunkach.

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

Należy skorzystać z następujących wzorów:

$$\begin{array}{ll} \text{min} & -(2^{I-1}) \\ \text{max} & 2^{I-1}-2^{-Q} \\ \text{rozd.} & 2^{-Q} \end{array}$$

Zatem:

$$\begin{array}{ll} \text{min} & -(2^2) \\ \text{max} & 2^2-2^{-9} \\ \text{rozd.} & 2^{-9} \end{array}$$

Po co i jak stosuje się zaokrąglenie wyniku?

Zaokrąglenie wyniku stosuje się w przypadku zmiennego przecinka pojedynczej precyzji. Przykładowo, dodając $x=10.0$ oraz $y = 0.000000238$ otrzymalibyśmy wynik 10,000000238, którego nie reprezentuje zmienny przecinek pojedynczej precyzji:

0x412000000 = 10.000000000
10.000000238 nie istnieje
0x412000001 = 10.000000950

Stąd właściwym działaniem jest zaokrąglenie wyniku w dół do 10.000000000. Alternatywą dla dokładnego obsłużenia liczb o takiej precyzji jest zastosowanie arytmetyki IQ Math.

Zaokrąglenie stosuje się również zazwyczaj dla wyniku ostatniej operacji. Można je uzyskać w niektórych rozkazach (przez automatyczne dodanie 8000h do akumulatora). Zaokrąglenie liczby x do najbliższej liczby całkowitej przebiega następująco: do x dodaj 0.50, a otrzymany rezultat zaokrąglij w dół do najbliższej liczby całkowitej.

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

SXM – Sign eXtention Mode – jest to bit w rejestrze ST1. Stan tego bitu kontroluje, czy rozszerzenie znakowe jest wykonywane czy też nie podczas ładowania operandów do akumulatorów lub na wejścia ALU. Zasada działania SXM jest następująca:

SXM = 1 → liczby ujemne dopełniane są na starszych bitach jedynkami, zaś dodatnie zerami

SXM = 0 → brak dopełnienia znakowego, starsze bity dopełniane są zerami zarówno dla liczb dodatnich, jak i ujemnych

Jest to mechanizm niezwykle istotny ze względu na to, że umożliwia mnożenie z zachowaniem znaku. Obsługa rozszerzenia znakowego wygląda następująco:

SSBX SXM ; sign-extension mode ON

RSBX SXM ; sign-extension mode OFF

Co to jest Saturation on Store (SST)?

Saturation on Store (SST) - jest to nasycanie przy zapamiętaniu. Gdy SST=1, włączone jest nasycanie wartości z akumulatora przed odesłaniem do pamięci. Nasycanie jest wykonywane po operacji przesunięcia. Podczas użycia SST wykonywane są następujące operacje:

- 40-bitowa wartość jest przesuwana (w prawo lub lewo) w zależności od instrukcji).
- 40-bitowa wartość jest nasycana do wartości 32-bitowej. Nasycenie zależy od bitu SXM.
 - Jeśli SXM = 0, generowana jest następująca 32-bitowa wartość:
 - FFFF FFFFh, jeśli wartość jest większa niż FFFF FFFFh
 - Jeśli SXM = 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
- Wartość jest przesyłana do pamięci w sposób zależny od instrukcji
- Zawartość akumulatora pozostaje niezmienna podczas operacji, podobnie jak OVx

Co to jest Overflow Mode i co zmienia w pracy procesora jego włączenie?

Overflow Mode jest trybem nadzoru przepełnienia zakresu. Włącza się go / resetuje poprzez modyfikację bitu OVM znajdującym się w rejestrze ST1. OVM determinuje, jaka wartość jest ładowana do akumulatora gdy dojdzie do przepełnienia:

gdy OVM = 0, przepełniony rezultat podlega przepełnieniu w akumulatorze

gdy OVM = 1, akumulator jest ustawiany na największą dopuszczalną wartość (7FFF FFFFh) lub na najmniejszą dopuszczalną wartość (8000 0000h) w razie gdy napotka wartość przekraczającą zakres. W związku z tym wynik obliczeń nie przekracza 32-bitów przy przekroczeniu zakresu.

Jak włącza się w procesorach DSP tryb Overflow Mode?

Poprzez ustawienie bitu OVM:

SSBX OVM

Odpowiednio, bit OVM resetowany jest przez:

RSBX OVM

Czym się różni przepełnienie od nasycenia?

Przy nasyceniu podczas przekroczenia zakresu w akumulatorze ustawiana jest maksymalna lub minimalna możliwa wartość, natomiast przepełnienie powoduje „przekręcenie” się wartości, polegające na tym, że bardzo duża wartość dodatnia może się stać bardzo dużą wartością ujemną, i odwrotnie.