Programowanie w C++

Opis procesora *Sextium*

materiały dydaktyczne udostępnione przez
Tomasza Wierzbickiego

1 Opis procesora Sextium II

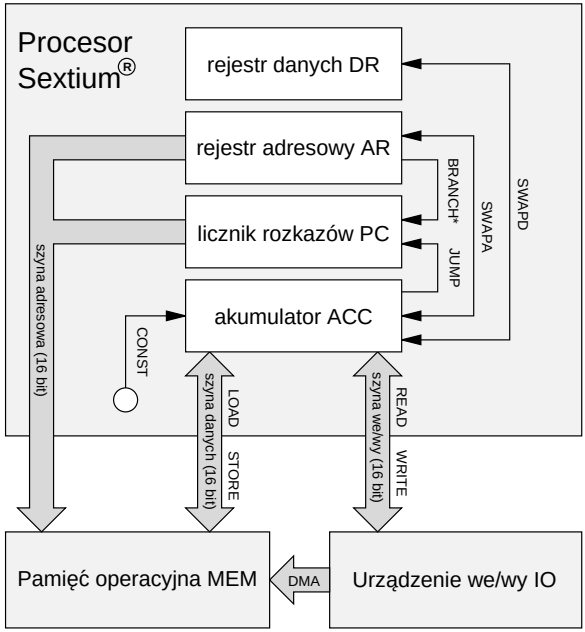
Budowa procesora Sextium II[1] o architekturze typu RISC[2] jest przedstawiona na rysunku 1. Poprzez szynę danych i szynę adresową do procesora jest podłączona pamięć (MEM) złożona z 64k słów 16-bitowych, a poprzez szynę wejścia/wyjścia — urządzenie wejścia/wyjścia (IO). Szyny: danych, adresowa i we/wy są 16-bitowe, podobnie jak rejestry: akumulator (ACC), rejestr danych (DR), rejestr adresowy (AR) i licznik rozkazów (PC). Rozkazy (jest ich 16) zajmują po 4 bity i są pakowane po 4 w słowie maszyny. Opis rozkazów jest przedstawiony w tablicach 1 i 2, a sposób pakowania rozkazów w słowie na rysunku 2.

Do przedstawiania zawartości pamięci i rejestrów procesora będziemy konsekwentnie używać zapisu szesnastkowego. Zawartość pojedynczego słowa będzie zatem opisana ciągiem czterech cyfr szesnastkowych. Dla wygody rozkazy mają swoje nazwy mnemoniczne. Zatem np. zamiast pisać „rozkaz E", będziemy pisać „rozkaz DIV". Dla procesora jednak rozkaz, to po prostu cztery bity (które my zapisujemy za pomocą pojedynczej cyfry szesnastkowej).

Cykl rozkazowy procesora polega na pobieraniu, dekodowaniu i wykonywaniu rozkazów. Wczytane słowo zawierające cztery rozkazy jest przechowywane w specjalnym *buforze rozkazów* (nie uwidocznionym na rysunku 1). W celu pobrania rozkazów procesor odwołuje się do pamięci przeważnie raz na cztery cykle rozkazowe, chyba że wykonuje instrukcję skoku. Wówczas bufor rozkazów jest opróżniany. Można zatem skoczyć jedynie do

[1] Pierwsza wersja procesora Sextium, o czym przekonali się studenci rocznika 1999/2000, była nieudana i została wkrótce zastąpiona modelem Sextium II.

[2] *Reduced instruction set computer*, procesor o ograniczonym zbiorze instrukcji: procesor w którym lista rozkazów jest maksymalnie skrócona. Dzięki temu rozkazy mogą być wykonywane bardzo szybko i sprawnie. Procesory RISC (posiadające kilkanaście do kilkudziesięciu rozkazów) są przy tej samej częstotliwości zegara kilkakrotnie bardziej efektywne niż procesory CISC (*complex instruction set computer*). Wynika to z faktu, że jeden cykl rozkazowy w takim procesorze trwa tylko jeden lub najwyżej kilka taktów zegara a nie kilkanaście czy kilkadziesiąt jak w procesorach typu CISC. Rozbudowana lista rozkazów może pomóc zoptymalizować kod wynikowy, jeśli programista pisze program w asemblerze, natomiast zbudowanie kompilatora języka wysokiego poziomu, który wykorzystywałby wiele rozkazów jest kłopotliwe. W praktyce kompilatory używają tylko najprostszych rozkazów procesora CISC i dodatkowe możliwości takiego procesora związane z bardzo bogatą listą instrukcji nie są wykorzystywane (optymalizacja kodu jest trudna). Tworzenie kompilatorów generujących efektywny kod dla procesorów RISC jest łatwiejsze niż dla procesorów CISC (efektywność jest osiągnięta dzięki masowości a nie różnorodności).



ACC, *accumulator*, akumulator: rejestr procesora, na którym są wykonywane operacje arytmetyczne i przesyłu danych.

DR, *data register*, rejestr danych: pomocniczy rejestr przechowujący drugi argument operacji arytmetycznych.

AR, *address register*, rejestr adresowy: rejestr, którego zawartość służy do adresowania pamięci w celu pobrania (zapisu) danych z (do) pamięci.

PC, *program counter*, licznik rozkazów: rejestr, którego zawartość służy do adresowania pamięci w celu pobrania kolejnych rozkazów do wykonania.

MEM, *memory*, pamięć operacyjna.

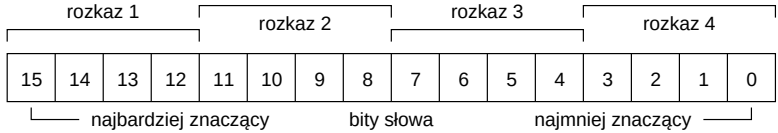
IO, *input/output*, urządzenie wejścia/wyjścia.

DMA, *direct memory access*, możliwość bezpośredniego zapisu do pamięci wprost z urządzenia wejścia/wyjścia bez udziału procesora.

Rysunek 1: Architektura procesora Sextium II

rozkaz	symb.	opis	uwagi
0	NOP	nic nie rób	przejdź do wykonania następnego rozkazu
1	LOAD	MEM[AR] $\rightarrow$ ACC	załaduj słowo o adresie znajdującym się w rejestrze adresowym do akumulatora
2	STORE	ACC $\rightarrow$ MEM[AR]	zapisz zawartość akumulatora do słowa o adresie znajdującym się w rejestrze adresowym
3	READ	IO $\rightarrow$ ACC	wczytaj słowo z urządzenia wejściowego do akumulatora
4	WRITE	ACC $\rightarrow$ IO	prześlij zawartość akumulatora do urządzenia wyjściowego
5	SWAPA	ACC $\leftrightarrow$ AR	zamień miejscami zawartość rejestru adresowego i akumulatora
6	SWAPD	ACC $\leftrightarrow$ DR	zamień miejscami zawartość rejestru danych i akumulatora
7	JUMP	ACC $\rightarrow$ PC	wpisz do licznika instrukcji zawartość akumulatora (skocz do instrukcji o adresie znajdującym się w akumulatorze)
8	BRANCHZ	if ACC $= 0$ then AR $\rightarrow$ PC	jeżeli akumulator zawiera liczbę 0, zapisz do licznika instrukcji zawartość rejestru adresowego (skocz do instrukcji o adresie znajdującym się w rejestrze adresowym)
9	BRANCHP	if ACC > 0 then AR $\rightarrow$ PC	jeżeli akumulator zawiera liczbę dodatnią, zapisz do licznika instrukcji zawartość rejestru adresowego (skocz do instrukcji o adresie znajdującym się w rejestrze adresowym)

Tablica 1: Zestaw rozkazów procesora Sextium II (część 1)



Rysunek 2: Sposób pakowania rozkazów procesora Sextium II w słowie maszynowym

rozkazu, który zajmuje cztery najbardziej znaczące bity słowa (jeśli sprawia to jakieś problemy, można poprzednie słowo wypełnić rozkazami NOP w celu wyrównania wskazanego rozkazu do granicy słowa). Rozkaz CONST służy do załadowania stałej do akumulatora. Stała jest umieszczana w słowie następującym bezpośrednio po słowie zawierającym ten rozkaz (lub w następnych słowach, jeżeli w jednym słowie znajduje się więcej niż jeden rozkaz CONST). Np. następująca para słów: 6AB0 0001 zawiera ciąg rozkazów SWAPD, CONST, ADD, NOP, po którym następuje liczba 1. Wykonanie tych rozkazów powoduje zwiększenie zawartości akumulatora o jeden. Podobnie ciąg sześciu słów:

A6AD 0004 0005 6A56 FFFA 2000

który symbolicznie zapiszemy jako:

CONST SWAPD CONST MUL
0004
0005
SWAPD CONST SWAPA SWAPD
FFFA
STORE NOP NOP NOP

powoduje załadowanie do rejestru danych liczby 4, do akumulatora liczby 5, przemnożenie zawartości akumulatora przez zawartość rejestru danych i zapisanie wyniku (liczby 20) do komórki pamięci o adresie FFFA.

Program dla procesora jest wpisywany do pamięci wprost z urządzenia wejścia/wyjścia (poprzez tzw. szynę DMA). W tym czasie procesor nie pracuje. Następnie urządzenie wejścia/wyjścia wysyła sygnał do procesora, który zeruje zawartość wszystkich swoich rejestrów i opróżnia bufor rozkazów, a następnie rozpoczyna wykonywanie cyklu rozkazowego. Procesor przekazuje sterowanie na powrót do urządzenia wejścia/wyjścia po wykonaniu rozkazu HALT lub w razie próby dzielenia przez zero. Pracę procesora może również przerwać urządzenie wejścia/wyjścia. Po uruchomieniu licznik rozkazów ma wartość 0000, zatem procesor rozpoczyna działanie od wykonania rozkazów znajdujących się w zerowym słowie pamięci. Algorytm pracy procesora jest opisany w tablicy 3, w której A++ oznacza zwiększenie zawartości rejestru A o jeden.

Słowa w operacjach arytmetycznych reprezentują liczby całkowite ze znakiem z przedziału $-2^{15} \div (2^{15} - 1)$ w zapisie uzupełnieniowym do dwóch, tj. zmiana znaku liczby polega na zanegowaniu wszystkich bitów liczby i dodaniu jedynki. Słowa od 0000 do 7FFF reprezentują liczby nieujemne 0 do 32767, słowa 8000 do FFFF zaś liczby -32768 do -1.

rozkaz	symb.	opis	uwagi
A	CONST	MEM[PC++] $\rightarrow$ ACC	zapisz słowo znajdujące się w komórce pamięci o adresie wskazywanym przez bieżącą zawartość licznika rozkazów do akumulatora
B	ADD	ACC + DR $\rightarrow$ ACC	dodaj do akumulatora zawartość rejestru danych
C	SUB	ACC – DR $\rightarrow$ ACC	odejmij od akumulatora zawartość rejestru danych
D	MUL	ACC $\times$ DR $\rightarrow$ ACC	pomnóż zawartość akumulatora przez zawartość rejestru danych
E	DIV	ACC / DR $\rightarrow$ ACC	podziel zawartość akumulatora przez zawartość rejestru danych
F	HALT	zatrzymaj	przekaż sterowanie do urządzenia we/wy

Tablica 2: Zestaw rozkazów procesora Sextium II (dokończenie)

2 Asembler

Zapisywanie programu wprost w kodzie maszynowym jest kłopotliwe. Asembler, inaczej *język adresów symbolicznych,* pozwala na symboliczne zapisywanie rozkazów i adresów uwalniając programistę od żmudnego i mechanicznego kodowania programu. Jest to przy tym język niskiego poziomu, w którym programista ma pełną kontrolę nad postacią kodu wynikowego. Zwykle jedna instrukcja asemblera jest przekładana na jeden rozkaz maszyny. Program w asemblerze jest dokładnym opisem kodu maszynowego, w którym rozkazy są reprezentowane przez ich nazwy mnemoniczne a adresy przez identyfikatory. Program tłumaczący (również zwany asemblerem) składa (ang. *assemble*) kod maszynowy według tego opisu, wstawiając kody rozkazów w miejsce ich nazw mnemonicznych, a wyliczone adresy w miejsce identyfikatorów. W językach wysokiego poziomu, takich jak np. SML, progamista posługuje się zwykle pojęciem *maszyny abstrakcyjnej*, której budowa jest nieraz bardzo odległa od architektury rzeczywistego procesora, na którym jest wykonywany skompilowany program (np. w SML-u dołączenie głowy do listy jest traktowane jako pojedyncza operacja, tymczasem przekłada się ono na ciąg wielu rozkazów procesora). Natomiast programując w asemblerze programista nadal myśli w kategoriach rzeczywistego procesora. Ponieważ asembler jest ściśle związany z architekturą danego procesora, w odróżnieniu od języków wysokiego poziomu, asembler jest zależny od maszyny. Dlatego nie istnieje jeden język

wyzeruj rejestry ACC, PC, AR i DR i opróżnij bufor rozkazów
loop
  if bufor rozkazów jest pusty **then**
    pobierz słowo z pamięci o adresie zawartym w PC
    PC++
  end if
  $R \leftarrow$ odkoduj następny rozkaz
  if $R = \mathrm{LOAD}, \mathrm{STORE}, \mathrm{READ}, \mathrm{WRITE}, \mathrm{SWAPA}, \mathrm{SWAPD}, \mathrm{ADD}, \mathrm{SUB}, \mathrm{MUL}, \mathrm{DIV}$ **then**
    wykonaj tę operację zgodnie z opisem w tablicach 1 i 2
  else if $R = \mathrm{CONST}$ **then**
    załaduj słowo z pamięci o adresie zawartym w PC do ACC
    PC++
  else if $R = \mathrm{JUMP}$ **then**
    zapisz do PC zawartość ACC
    opróżnij bufor rozkazów
  else if ($R = \mathrm{BRANCHZ}$ i $\mathrm{ACC} = 0$) lub ($R = \mathrm{BRANCHP}$ i $\mathrm{ACC} > 0$) **then**
    zapisz do PC zawartość AR
    opróżnij bufor rozkazów
  else if $R = \mathrm{HALT}$ **then**
    zatrzymaj pracę
  end if
end loop

Tablica 3: Cykl rozkazowy procesora Sextium II

zwany asemblerem, są tylko asemblery konkretnych procesorów. Aby język programowania był niezależny od procesora, jego maszyna abstrakcyjna musi być idealizacją, częścią wspólną wszystkich procesorów, na które programy w tym języku mają być kompilowane. Przykładem takiego języka jest C, zwany niekiedy *asemblerem strukturalnym*. Należy on już jednak do kolejnego piętra w hierarchii „poziomu" języków programowania.

2.1 Asembler procesora Sextium II

Jednostkami leksykalnymi asemblera procesora Sextium II są *literały całkowitoliczbowe*, tj. niepuste ciągi cyfr dziesiętnych, poprzedzone opcjonalnie znakiem minus, np. 73, *literały szesnastkowe*, tj. ciągi dokładnie czterech cyfr szesnastkowych poprzedzone znakami 0x lub 0X, np. 0xFAFA, *słowa kluczowe*:

ADD BRANCHP BRANCHZ CONST DATA DIV HALT JUMP LOAD MUL
READ STORE SUB SWAPA SWAPD WRITE

identyfikatory, tj. ciągi liter i cyfr zaczynające się literą i różne od słów kluczowych oraz *symbol pomocniczy* „:". Identyfikatory są używane jako *etykiety*. Wielkie i małe litery są

rozróżniane w identyfikatorach i słowach kluczowych, natomiast cyfry szesnastkowe A–F można pisać zarówno wielką, jak i małą literą. Znakiem początku komentarza jest #. Komentarz rozciąga się do końca wiersza w którym występuje (do znaku nowego wiersza).

Program w asemblerze jest ciągiem *instrukcji asemblera*, po co najwyżej jednej w wierszu. Instrukcja może się składać z pojedynczego słowa kluczowego

ADD BRANCHP BRANCHZ DATA DIV HALT JUMP LOAD MUL READ STORE SUB SWAPA SWAPD WRITE

lub może być postaci

CONST *parametr*

gdzie *parametr* to albo literał dziesiętny lub szesnastkowy, albo identyfikator (etykieta). Każda instrukcja może być opcjonalnie poprzedzona napisem postaci

etykieta :

Zauważmy że rozkaz procesora NOP *nie jest* instrukcją asemblera, natomiast instrukcja DATA nie ma odpowiednika wśród rozkazów procesora. Identyfikatory (etykiety) są symbolicznymi reprezentacjami adresów pamięci. Każda etykieta powinna pojawić się dokładnie raz w programie przed dwukropkiem i może pojawiać się wielokrotnie w instrukcji CONST. Zadaniem asemblera jest przetłumaczenie symbolicznych nazw instrukcji na odpowiadające im rozkazy, spakowanie rozkazów po cztery w 16-bitowe słowa (wypełniając puste miejsca rozkazami NOP), wyznaczenie faktycznych adresów instrukcji opatrzonych etykietami i wygenerowanie kodu binarnego. Asembler generuje rozkazy w takiej kolejności, w jakiej odpowiednie instrukcje znajdują się w tekście programu. Instrukcja CONST powoduje wygenerowanie rozkazu ładującego stałą opisaną podanym literałem lub etykietą do akumulatora. Instrukcja DATA powoduje zarezerwowanie słowa w pamięci w miejscu, w którym występuje (dlatego zwykle umieszcza się ją na końcu programu). Do tego słowa w pamięci wpisywana jest liczba 0x0000 (tj. pamięć dla zmiennych jest inicjowana podczas uruchamiania programu). Do adresu tego słowa można się odwoływać poprzez etykietę tej instrukcji (zatem użycie rozkazu DATA bez etykiety ma niewiele sensu). Program tłumaczący program w asemblerze na kod wynikowy jest dwuprzebiegowy: w pierwszym przebiegu tworzy się kod wynikowy „na próbę”, nie wstawiając do niego stałych opisanych przez etykiety (nie są one bowiem jeszcze znane). W tym przebiegu ustala się ich wartości. W drugim przebiegu generuje się ostatecznie kod wynikowy, ponieważ wartości etykiet są już ustalone. Tablica 6 zawiera program w asemblerze obliczający największy wspólny podzielnik dwóch liczb według algorytmu z tablicy 4, tablica 7 jego tłumaczenie na kod wynikowy, a tablica 5 plik z kodem wynikowym w postaci szesnastkowej.

wczytaj x
wczytaj y
while $y \neq 0$ **do**
  $z \leftarrow x \bmod y$
  $x \leftarrow y$
  $y \leftarrow z$
end while
wypisz x

Tablica 4: Algorytm Euklidesa obliczania największego wspólnego podzielnika

A532 001C A532 001D 9516 001D A568 0019
6A51 001C ED6A 001C 51C6 A562 001E A516
001D A562 001C A516 001E A562 001D A700
0004 001C F000 0000 0000 0000

Tablica 5: Plik gcd.hex zawierający program dla procesora Sextium II w postaci szesnastkowej obliczający największy wspólny podzielnik dwóch liczb (opis w tablicy 7)

```
# Program gcd.asm
# Wczytuje dwie liczby i wypisuje
# ich największy wspólny dzielnik
#
        CONST x         # czytaj x
        SWAPA
        READ
        STORE
        CONST y         # czytaj y
        SWAPA
        READ
        STORE
dalej:  CONST y         # czy y=0?
        SWAPA
        LOAD
        SWAPD
        CONST koniec
        SWAPA
        SWAPD
        BRANCHZ
        SWAPD           # z=x/y
        CONST x
        SWAPA
        LOAD
        DIV
        MUL             # z=z*y
        SWAPD           # z=x-z
        CONST x
        SWAPA
        LOAD
        SUB
        SWAPD
        CONST z
        SWAPA
        SWAPD
        STORE
        CONST y         # x=y
        SWAPA
        LOAD
        SWAPD
        CONST x
        SWAPA
        SWAPD
        STORE
        CONST z         # y=z
        SWAPA
        LOAD
        SWAPD
        CONST y
        SWAPA
        SWAPD
        STORE
        CONST dalej
        JUMP
koniec: CONST x         # pisz x
        SWAPA
        LOAD
        WRITE
        HALT
     x: DATA            # zmienne
     y: DATA
     z: DATA
```

Tablica 6: Program w asemblerze obliczający największy wspólny podzielnik dwóch liczb

adres	zawartość słowa	opis zawartości słowa
0000	A532	CONST SWAPA READ STORE
0001	001C	adres reprezentowany przez zmienną x
0002	A532	CONST SWAPA READ STORE
0003	001D	adres reprezentowany przez zmienną y
dalej = 0004	9516	CONST SWAPA LOAD SWAPD
0005	001D	adres reprezentowany przez zmienną y
0006	A568	CONST SWAPA SWAPD BRANCHZ
0007	0019	adres reprezentowany przez zmienną koniec
0008	6A51	SWAPD CONST SWAPA LOAD
0009	001C	adres reprezentowany przez zmienną x
000A	ED6A	DIV MUL SWAPD CONST
000B	001C	adres reprezentowany przez zmienną x
000C	51C6	SWAPA LOAD SUB SWAPD
000D	A562	CONST SWAPA SWAPD STORE
000E	001E	adres reprezentowany przez zmienną z
000F	A516	CONST SWAPA LOAD SWAPD
0010	001D	adres reprezentowany przez zmienną y
0011	A562	CONST SWAPA SWAPD STORE
0012	001C	adres reprezentowany przez zmienną x
0013	A516	CONST SWAPA LOAD SWAPD
0014	001E	adres reprezentowany przez zmienną z
0015	A562	CONST SWAPA SWAPD STORE
0016	001D	adres reprezentowany przez zmienną y
0017	A700	CONST JUMP NOP NOP
0018	0004	adres reprezentowany przez zmienną dalej
koniec = 0019	A514	CONST SWAPA LOAD WRITE
001A	001C	adres reprezentowany przez zmienną x
001B	F000	HALT NOP NOP NOP
x = 001C	0000	miejsce przechowywania zmiennej x
y = 001D	0000	miejsce przechowywania zmiennej y
z = 001E	0000	miejsce przechowywania zmiennej z

Tablica 7: Tłumaczenie programu w asemblerze z tablicy 6 na kod procesora Sextium II