PANEVROPSKI UNIVERZITET APEIRON

FAKULTET INFORMACIONIH TEHNOLOGIJA

BANJA LUKA

Seminarski rad

MEMORIJSKI REGISTRI U IA-32 ARHITEKTURI

Mentor:

Mr Igor Dugonjić

Student:

Siniša Božić, 192-20/RITP

Banja Luka, 2021.

SADRŽAJ

UVOD

Način organizacije memorijskog prostora je zavisan od date arhitekture procesora. U principu, temelji modernog procesorskog dizajna su udareni krajem 1960ih i naročito početkom 1970ih godina. Sa početka se radilo o 4-bitnoj i 8-bitnoj arhitekturi koja se inkrementalno nadograđivala do aktuelne 64-bitne. Iako je riječ o vremenskom periodu od 50 godina, osnovni postulati organizacije procesorske jedinice su i dalje aktuelni i ko ima potrebu da se bavi arhitekturom procesora kao takvog, bilo sa hardverskog ili softverskog nivoa, treba da poznaje njegove osnovne gradivne elemente.

U radu se posmatra i analizira memorijska struktura IA32 (ili x86) arhitekture koja je inicijalno predstavljena 1985. godine od strane Intel korporacije i bila je aktuelna sve do 1999. godine kada ju je zamijenila, tačnije rečeno kada je nadograđena sa x86-64 arhitekturom. Imajući u vidu memorijska ograničenja IA-32 arhitekture, može se tvrditi da će ona još biti aktuelna u budućnosti, što opravdava proučavanje i organizaciju memorijskog prostora na istoj. Sa softverske strane, budući da su niži programski jezici u stanju da direktno adresiraju i manipulišu memorijskim prostorom, a imajući u vidu da je asemblerski jezik hijerarhijski gledano prvi čovjeku 'smislen' jezik[1], razumijevanje memorijske organizacije je ključno za ispravno pisanje asemblerskog koda, a poznavanje asemblerskog koda pruža uvid u strukturu procesa koji se dešavaju pri izvršavanju instrukcija. Što se više ide po navedenoj hijerarhiji programskih jezika, nivoi apstrakcije su sve viši i sve više je zamagljenije šta tačno koja naredba čini. Što se tiče samih memorijskih registara, u literaturi ima manjih razlika u pogledu njihovih klasifikacija, jer IA-32 arhitektura obuhvata nekoliko tipova procesora kao što su 80386, 80486 te Pentium procesori, ali u ovom radu će se te razlike zanemariti.

Cilj rada je dakle razumijevanje strukture memorijskih registara u IA-32 procesorskoj arhitekturi naročito u pogledu razumijevanja i pisanja programa u asemblerskom programskom jeziku, obzirom da je programski kod asemblerskog programskog jezika zavisan od procesorske arhitekture za koju je napisan odnosno na kojoj se izvršava.

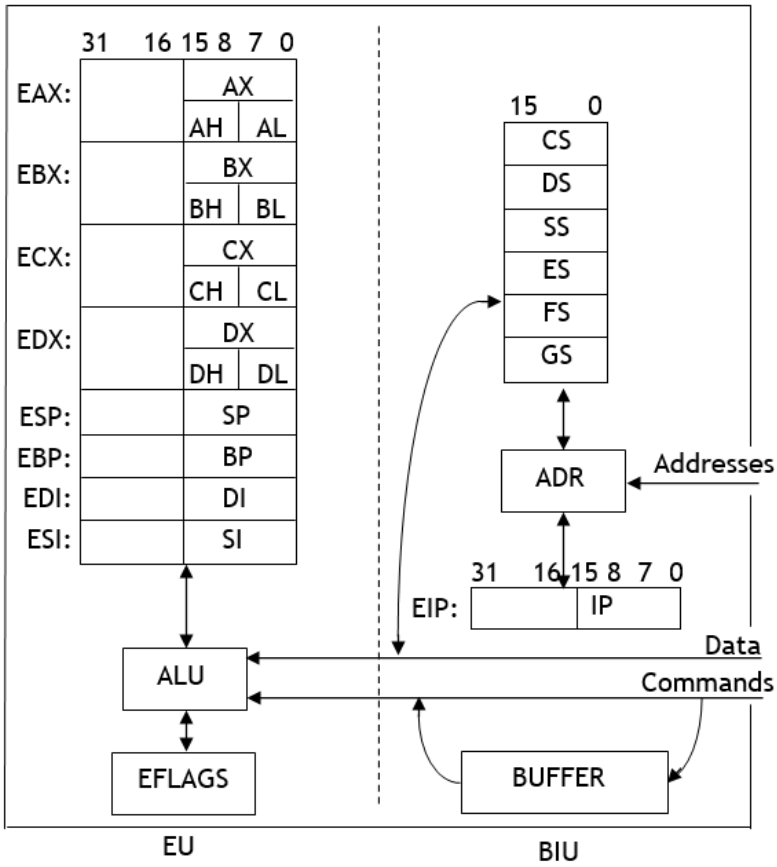
[1] Na primjer, instrukcija za sabiranje registara EAX i EBX i rezultat ponovo smjesti u EAX se kodira u mašinskom jeziku na idući način: 03 C3 , što je čovjeku potpuno neprilagođeno tj. nerazumljivo.

1 IA-32 STRUKTURA

IA-32 procesor ima dve glavne komponente:

- Izvršna jedinica (Executive Unit), koja pomoću ALU jedinice (Arithmetic and Logic Unit) izvršava mašinske instrukcije,
- BIU jedinica (Bus Interface Unit), koja priprema izvršenje svake mašinske stukture. Ona čita instrukciju iz memorije, dekodira je i proračunava memorijski adresni prostor operanda ako on postoji. Izlaz se sprema u 15-bajtovni bafer, odakle se prosljeđuje ka izvršnoj jedinici.

Izvršna i BIU jedinica rade paralelno – dok izvršna jedinica izvršava trenutnu instrukciju, BIU priprema iduću. Ove aktivnosti su sinhronizovane, ona jedinica koja prije završi trenutni zadatak čeka drugu jedinicu. U pitanju je tzv 'pipelining', koncept koji je Intel primjenio još od 8088/8086 arhitekture i koji ubrzava obradu informacija.



Slika 1. Prikaz strukture IA-32 procesora

1.1 SKUP STANDARDNIH INSTRUKCIJA

Skup standardnih instrukcija obuhvata:

1. Instrukcije prenosa,
2. Aritmetičke instrukcije,
3. Logičke instrukcije,
4. Instrukcije pomjeranja,
5. Instrukcije skoka.

1. Instrukcije prenosa:
   MOV a,b – prebacivanje podatka iz jedne lokacije u drugu,
   IN a,b – prebacivanje iz registra kontrolora periferije a u registar procesora b
   OUT a,b – prebacivanje iz registra procesora a u registar kontrolora periferije b
   LOAD a – prebacivanje podatka sa lokacije a u akumulator
   STORE a – smještanje podatka iz akumulatora na lokaciju a
   PUSH a – podatak sa lokacije a se smješta na stek
   POP a – podatak sa steka se smješta na lokaciju a

2. Aritmetičke instrukcije:
   ADD a,b,c – sabiranje operanada sa lokacija a i b i smještanje na lokaciju c
   ADD a,b – sabiranje operanada a na lokaciju b
   ADD a – izvor i odredište je akumulator a
   ADD – vrh steka je implicitan izvor i odredište
   SUB a,b,c ili SUB a,b, ili SUB a, ili SUB – oduzimanje
   MUL abc, ili MUL a,b, ili MUL a, ili MUL – množenje
   DIV abc, ili DIV a,b, ili DIV a, ili DIV – dijeljenje
   INC a,b – inkrementiranje operanda sa lokacije a i smještanje na lokaciju B, alternativne komande INC a, INC
   DEC a,b ili DEC a, ili DEC – dekrementiranje

3. Logičke instrukcije:
   AND a,b,c ili AND a,b, ili AND a, ili AND – operacija logičko I

OR a,b,c, ili OR a,b, OR a, OR – operacija logičko ILI

XOR a,b,c ili XOR a,b, ili XOR a, ili XOR – operacija logično XOR

NOT a,b, ili NOT a, ili NOT – operacija negacije

4. Instrukcije pomjeranja:

   ASR a,b ili ASR a, ili ASR – aritmetičko pomjeranje udesno, na mjesto novog bita se upisuje stari bit

   ASL a,b ili ASL a ili ASL – aritmetičko pomjeranje ulijevo, na mjesto novog bita se upisuje 0

   LSR a,b ili LSR a ili LSR – logičko pomjeranje udesno, na mjesto novog bita se upisuje 0

   LSL a,b ili LSL a ili LSL – logičko pomjeranje ulijevo, na mjesto novog bita se upisuje 0

   ROR, ROL, RORC, ROLC – instrukcije rotiranja (bez ili sa prenosa).

5. Instrukcije skoka:

   JMP a – bezuslovni skok na adresu a koja se upisuje u programski brojač procesora kao naredna instrukcija za izvršavanje

   JMPIND a – bezuslovni skok na adresu koja je specificirana operandom a

   BEQL a, BNEQ a, BGRTU a, BGREU a, BLSSU a, BLEQU a, BGRT a, BGRE a, BLSS a, BLEQ a, BNEG a, NBBG a, BOVF a, BNVF a – instrukcije uslovnog skoka u zavisnosti od vrijednosti indikatora, a je pomjeraj u odnosu na programski brojač na koji treba skočiti

   JEQL a, JNEQ a, JGRTU a, JGREU a, JLSSU a, JLEQU a, JGRT a, JGRE a, JLLS a, JLEQ a, JNEG a, JNNG a, JOVF a, JNVF a – instrukcije bezuslovnog skoka u zavisnosti od programske statusne riječi, a je adresa na koju se skače,

   JSR a – skok na potprogram na adresi a

   RTS – povratak iz potprograma

   INT a – skok na prekidnu rutinu na adresi a

   RTI – povratak iz prekidne rutine.

1.2 MEMORIJSKI REGISTRI

Memorijski registri predstavljaju brzu memoriju koja je, kako se vidi iz Priloga 1 dio arhitekture samog procesora. Svrha memorijskih registara je ubrzavanje protoka podataka i interakcije izvršne i BIU jedinice, budući da bi performanse sistema bile mnogo manje da centralna procesorska jedinica mora da doprema instrukcije i podatke iz sistemske RAM memorije.

Kada je u pitanju IA-32 arhitektura, govorimo o 8-bitnim, 16-bitnim i 32-bitnim registrima. Njihova uobičajena podjela je na:

- Opšte registre,
- Kontrolne registre,
- Segmentne registre.

1.2.1. Opšti registri

Opšti registri obuhvataju:

- Registre podataka (data register),
- Pokazivački registri (pointer register) i
- Indeksni registri (index register)

Registri podataka

IA-32 arhitektura koristi 32-bitne registre, i takvi registri su novina kod ove arhitekture. Prijašnje 8086 arhitekture su koristile primarno 4-bitne i 8-bitne registre[2]. U pitanju je veličina riječi, odnosno eng. 'Word size', na osnovu koje se izražavaju i mjere ostalih bitnih komponenti kao npr. širina magistrale i veličina instrukcije. Kuriozitet je da Windows API drugačije definiše riječ, kao 16-bitnu jedinicu, duplu riječ (DWORD) kao 32-bitnu te QWORD kao 64-bitnu.

[2] 8086 arhitektura je 8/16-bitna arhitektura koja je prethodila IA-32 arhitekturi. Svaki IA-32 procesor je 8086 kompatibilan.

Definisana su četiri 32-bitna registra, i to:

- EAX, akumulator, korišten od većine instrukcija kao operand,
- EBX, bazni registar,
- ECX, brojački registar, uglavnom korišten kao numerički gornji limit za instrukcije koje su repetitivne prirode,
- EDX, podatkovni ili data registar, korišten zajedno sa EAX registrom kad rezultat prevazilazi duplu riječ (32-bita).

Pobrojani registri se mogu koristiti i u manjim fragmentima, odnosno njihove donje polovine se mogu koristiti kao četiri 16-bitna registra (gornje polovine nemaju ime i nisu raspoložive):

- AX,
- BX,
- CX,
- DX

Pri tom, 16-bitni registri se također mogu koristiti kao kombinacije 8-bitnih: AH, BH, CH i DH su gornji, važniji 8-bitni registri (gornji dijelovi AX, BX, CX i DX registara), te AL, BL, CL i DL registri, kao niži dio 16-bitnih registara.

Pokazivački registri

Pokazivački registri su 32-bitni i to su stek registri. Stek je memorijsko područje koje funkcioniše po LIFO principu (zadnji ulazni sadržaj je prvi izlazni). To su:

- ESP, Stack Pointer, odnosi se na zadnji element stavljen na stek (element na vrhu steka),
- EBP, Base Pointere, prvi element stavljen na stek (baza steka).

Također važi podjela na 16-bitne registre, odnosno SP i BP registre.

Indeksni registri

U indeksne registre spadaju 32-bitni registri ESI i EDI, koji se uglavnom koriste za indeksirano adresiranje, te analogno ranijim tipovima registara, sadrže 16-bitne registre SI i DI.

Tabela 1: Pregled opštih registara

32-bitni registri		16-bitni registri		8-bitni registri	
EAX	EBP	AX	BP	AH	AL
EBX	ESI	BX	SI	BH	BL
ECX	EDI	CX	DI	CH	CL
EDX	ESP	DX	SP	DH	DL
Bitovi 16-31		Bitovi 8-15		Bitovi 0-7	

1.2.2 Kontrolni registri

Kontrolni registri (flag) su 1-bitni indikatori koji pokazuju status procesora ili kontrolu operacije procesora. Kod IA-32 arhitekture EFLAGS statusni registar je širine 32 bita ali samo 9 bita su korišteni.

Tabela 2: Pregled kontrolnih registara

31	30	...	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	...	x	OF	DF	IF	TF	SF	ZF	x	AF	x	PF	x	CF

CF (Carry Flag) je transportni i pokazuje da je izvršena instrukcija generisala prenos sa mjesta najveće težine (krajnji lijevi bit). Na primjer, kod sabiranja

1001 0011 + 147 +

0111 0011 115

1 0000 0110 262

CF je 1.

AF (Auxiliary Flag) pokazuje vrijednost transporta sa bita 3 na bit 4, u gornjem primjeru vrijednost je 0.

PF (Parity Flag) pokazuje da je instrukcija svojim radom generisala rezultat sa parnim brojem jedinica.

ZF (Zero Flag) iznosi 1 ako je rezultat instrukcije nule i obrnuto.

SF (Sign Flag) iznosi 1 ako je rezultat instrukcije negativan broj, u suprotnom je 0.

OF (Overflow Flag) pokazuje da je došlo do prekoračenja krajnjeg lijevog bita podatka nakon izvršene transakcije.

DF (Direction Flag) kontroliše smjer izvršenja instrukcija u radu sa nizovima. Kada je DF vrijednost jednaka 0, string operacija ima smjer izvršenja sa lijeva na desno, a ako je vrijednost DF jednaka 1 string operacija ima smjer izvršenja sa desna na lijevo.

IF (Interrupt Enable Flag) omogućava prekide. Ako je vrijednost 1 prekidi su dozvoljeni, ako je 0 prekidi neće biti dozvoljeni.

TF (Trap Flag) je debaging flag. Ako je vrijednost 1, nakon izvršenja svake instrukcije sistem se zaustavlja.

1.2.3 Segmentni registri

Kod 8086 procesora, segment može označavati:

1. Blok memorije određene veličine, nazvan fizičkim segmentom. Broj bajtova u fizičkom segmentu je 64 Kb za 16-bitne procesore i 4 Gb za 32-bitne procesore.
2. Varijabilni blok memorije, nazvan logičkim segmentom ispunjenim programskim kodom ili podacima.

IA-32 arhitektura poznaje 4 tipa segmenata:

- Code Segment (CS), koji sadrži instrukcije,
- Data Segment (DS), koji sadrži podatke kojima pristupaju instrukcije,
- Stack Segment (SS).

2 PRIMJERI PROGRAMSKOG KODA

2.1 Hello World

```
segment .text      ;segment za programski kod
   global _start    ;linker

_start:            ;start za linker
   mov edx,len     ;dužina poruke
   mov ecx,msg      ;tekst poruke (string)
   mov ebx,1       ;file descriptor (stdout)
   mov eax,4       ;system call number (sys_write)
   int 0x80        ;poziv kernela

   mov eax,1        ;system call number (sys_exit)
   int 0x80        ;poziv kernela

segment .data       ;podatkovni segment
msg     db 'Hello, world!',0xa   ;string
len     equ     $ - msg          ;dužina stringa
```

Ispis programa po izvršenju:

```
Hello, world!
```

2.2 Prikaz 9 zvjezdica

```
section .text            ;segment za programski kod
   global _start         ;linker

_start:          ;start za linker
   mov  edx,len  ;dužina poruke
   mov  ecx,msg  ;tekst poruke
   mov  ebx,1    ;file descriptor (stdout)
   mov  eax,4    ;system call number (sys_write)
   int  0x80     ;poziv kernela
```

```
   mov  edx,9    ;dužina poruke
   mov  ecx,s2   ;tekst poruke
   mov  ebx,1    ;file descriptor (stdout)
   mov  eax,4    ;system call number (sys_write)
   int  0x80     ;poziv kernela

   mov  eax,1    ;system call number (sys_exit)
   int  0x80     ;poziv kernela

section .data     ;podatkovni segment

msg db 'Prikaz 9 zvjezdica',0xa ;tekst poruke
len equ $ - msg  ;dužina poruke
s2 times 9 db '*'
```

Ispis programa po izvršenju:

```
Prikaz 9 zvjezdica
*********
```

U oba primjera se uočava korištenje komande MOV, koja ima sljedeću sintaksu:

```
MOV dest, src
```

odnosno

```
MOV  register, register

MOV  register, immediate

MOV  memory, immediate

MOV  register, memory

MOV  memory, register
```

Ova komanda direktno manipuliše sa memorijskim registrima. U navedenim promjerima je vidljivo da se koriste 32-bitni registri EAX, EBX, ECX i EDX.

Ako bi ova dva programska koda napisali u programskom jeziku Python 3, to bi izgledalo ovako:

- `print('Hello world!)`
- ```
  a='*'
  print (9*a)
  ```

Iz navedenog se jasno uočava da su niži programski jezici mnogo bliži hardveru na kome se izvršavaju, i da na tom nivou apstrakcije se barata direktno sa hardverskim komponentama kojima se pristupa putem odgovarajućih komandi asemblerskog jezika. S druge strane, moderni programski jezici automatizuju ove radnje niskog nivoa te programski jezici počinju u sve većoj mjeri da liče govornim jezicima.
  ```

ZAKLJUČAK

Memorijska arhitektura IA-32 je donijela novine u odnosu na stariju 8086 arhitekturu, u smislu predstavljanja 32-bitnih memorijskih registara kao i proširenje nekih 16-bitnih (FS, GS). Memorijski registri kao takvi predstavljaju brze memorijske komponente integrisane u sam mikroprocesor i njihova struktura i organizacija direktno utiče na performanse datog sistema. Opisali smo opštu strukturu IA-32 arhitekture i naveli njene najvažnije dijelove. Kako smo vidjeli, u ovoj arhitekturi postoje 32-bitni, 16-bitni i 8-bitni registri koji mogu na više načina da se klasificiraju. Pri tom, najvažnija je podjela na opšte, kontrolne i segmentne registre. Naveli smo i osnovne instrukcije koje se koriste u asemblerskom programskom jeziku i naveli dva jednostavna primjera programskog koda. Uporedili smo ove programe sa istovjetnim programskim kodom u jeziku Python 3.

Kao zaključak može se tvrditi da poznavanje strukture i organizacije memorijskih registara je preduslov za uspješno programiranje u asemblerskom programskom jeziku, jer je u pitanju jezik nižeg ranga koji direktno komunicira sa pojedinim hardverskim komponentama procesora. Kao takav, on je stepenicu više od mašinskog jezika koji je ljudima nerazumljiv. Višedecenijska evolucija u računarskoj nauci je dovela do razvoja i stvaranja programskih jezika više generacije, koji postaju sve sličniji ljudskom jeziku.

LITERATURA

[1] Assembly Language Tutorial, *Tutorialspoint.com*, posjećeno: 17.01.2021

[2] L.A. Bong, Overview of IA-32 assembly programming, *University of Tromsø*

[3] Predavanja sa predmeta Niži programski jezici i programski prevodioci, *Panevropski Univerzitet Apeiron, Fakultet Informacionih tehnologija*

[4] W. Stallings, Computer Organization and Architecture, *Pearson*, 2013