

|   |                                 |
|---|---------------------------------|
| <b>Технически Университет - Габрово</b>   | <b>Организация на компютъра</b> |
| <b>Тема: Микропроцесори – част 1</b>  | <b>Лабораторно № 1</b>          |
| <b>Цел:</b> Запознаване с вътрешната архитектура на виртуален микропроцесор. Изпълнение на инструкциите на ниво микрокод. |                                 |
|   |                                 |

## I. Теоретична част

Микропроцесорите (Central Point Unit - CPU) имат за цел изпълнението на програмен код. Този код може да е част от операционната система или приложна програма. Микропроцесорите обработват само целочислени данни. Производителността на микропроцесорите се измерва в брой инструкции, които се изпълняват за една секунда (Million Instructions Per Second - MIPS). Операциите с операнди с плаваща запетая се изпълняват от *копроцесор* (Floating Point Unit - FPU). Производителността на копроцесорите се измерва с броя операции с плаваща запетая, които се изпълняват за една секунда (Floating Point Operations Per Second - FLOPS). На настоящият етап микропроцесорът и копроцесорът са в един чип. Производителността на CPU зависи от множество фактори, най-важните от които са: вътрешна архитектура, честота на такта за синхронизация на работа на микропроцесора (CPU CLOCK) и разрядност на CPU.

Микропроцесорите извличат от RAM инструкциите, които трябва да изпълнят. Тези инструкции (програмен код) трябва да са записани в динамична (DRAM) или статична (SRAM) памет (кеш). Всеки микропроцесор съдържа процедури чрез които се изпълнява всяка инструкция. Тези процедури съдържат микрокод, който е специфичен за всеки процесор. Като минимум, микропроцесорите трябва да съдържат следните базови модули:

- Аритметично-логическо устройство (ALU). Неговата функция е изпълнението на всички аритметични и логически инструкции, както и инструкции за преместване.
- Регистри. Това е най-бързата памет (статична RAM), която се използва за запис на операндите и съхраняване на резултата. Колкото повече регистри съдържа един микропроцесор, толкова по-рядко ще се обръща към бавната, външна за него RAM. От разрядността на регистрите се определя разрядността на CPU. Ако регистрите са 32-битови, то микропроцесорът е 32 битов.
- Вътрешна шина. Комуникацията на информацията между основните модули на CPU се реализира чрез вътрешната шина на микропроцесора. Комуникацията се реализира синхронно на такта CPU CLOCK.
- Памет за съхранение на микрокода.

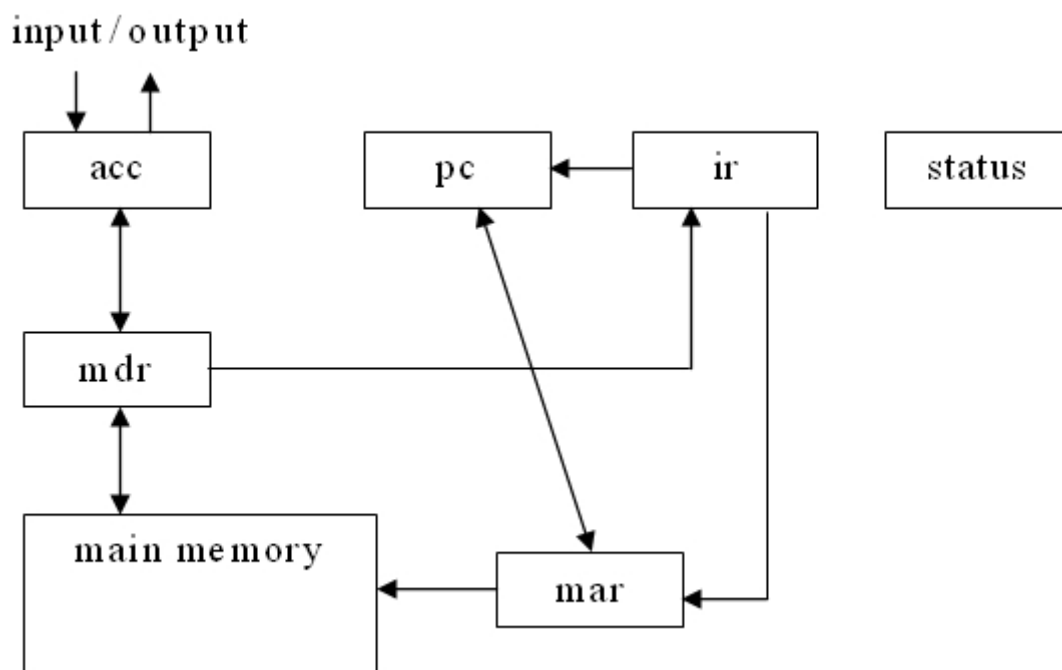
Изпълнението на инструкциите зависи от архитектурата на CPU. Ако в даден момент от време се получава резултата от изпълнение на няколко инструкции, то архитектурата е *паралелна*. Ако модул от CPU решава кои инструкции от потока с инструкции трябва да се изпълнят едновременно, то архитектурата е *паралелна на апаратно ниво*. Типичен пример за такава архитектура е супер-скаларната (Pentium II, Pentium III). Ако компилаторът решава кои инструкции трябва да се изпълнят едновременно, то архитектурата е *паралелна на програмно ниво*. Типичен пример за такава архитектура е Very Long Instruction Word (VLIW) - Itanium, Itanium 2. Има и архитектури, които са *псевдо-*

*паралелни*. При тях изпълнението на всяка инструкция се разбива на фази. За изпълнение на всяка фаза отговаря отделно устройство. Всички те могат да работят паралелно едно спрямо друго. Следователно, в даден момент такъв процесор може да обработва няколко инструкции, всяка от които се намира в различна фаза от своето изпълнение. Колкото повече е броят на фазите, толкова по-ефективен е микропроцесорът. Например, ако процесорът използва 16 фази за изпълнение на инструкциите, то в даден момент една инструкция връща резултат, а до 15 други са в различни фази от своето изпълнение. Типичен пример за такава архитектура е *скаларната (конвейерна)*.

## II. Виртуални микропроцесори

За целите на обучението се използват виртуални микропроцесори, които имат опростена вътрешна архитектура и набор от микрокод процедури. Целта е да се разбере как функционира CPU, а не как точно работи конкретен модел микропроцесор. Ще използваме симулатора CPUSim. Това е Java приложение, което позволява създаването на виртуални CPU с определена архитектура, набор от инструкции и микрокод процедури. По подразбиране се използва архитектура с име *Wombat*, която симулира работата на CPU с 6 регистъра, 12 инструкции и 128 байта RAM. Потребителят може да създава собствена CPU архитектура и да допълва набора от инструкции. Можете да създавате собствени програми, които да изпълнявате нацяло или на стъпки (инструкция след инструкция). Можете да наблюдавате последователността на изпълнение на кода на ниво инструкции или на ниво микрокод. Във всеки един момент от време се вижда как се променя съдържанието на регистрите и паметта в избрана бройна система.

Вътрешната архитектура на виртуалния CPU е показана на Фиг. 1.



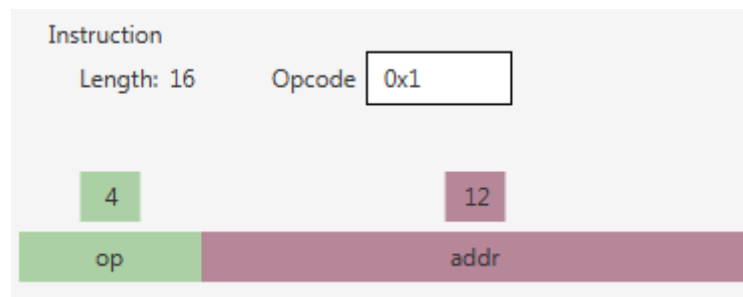
Фиг. 1. Архитектура на виртуалния CPU

Регистрите, които CPU използва, са следните:

- *acc* – Акумулатор (16 bits). Чрез акумулатора се въвежда и извежда информация. Той участва и при всички аритметичните операции.
- *pc* – програмен брояч (12 bits). Съдържа адреса на инструкцията, която предстои да се изпълни.
- *ir* – регистър на инструкциите (16 bits). В този регистър се зарежда кода на инструкцията, която подлежи на изпълнение. От операционния код се разбира какво е действието на инструкцията, какви данни изисква и къде се записва резултата.
- *mar* – регистър за адресиране на паметта (12 bits). Този регистър съдържа адреса на клетка от паметта от която ще се чете или в която ще се записва.
- *mdr* – регистър чрез който се достъпва паметта (16 bits). Данните, които се четат от клетка в паметта се записват в този регистър. Данните, които се записват в клетка от паметта, се записват в този регистър.
- *status* – регистър на състоянието (3 bits). Всеки бит от регистъра кодира конкретно състояние на процесора. Например бит 2 е HALT. При стойност 1 процесорът прекратява изпълнението на инструкции.

Виртуалният CPU (виртуална машина *Wombat*) изпълнява инструкциите в следната последователност:

1. Извличане на операционния код на инструкцията. На Фиг. 2 е показан форматът на инструкция *load*.



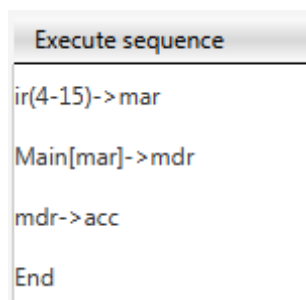
Фиг. 2. Формат на инструкция *load*

Тази инструкция зарежда акумулатора със съдържанието на клетка от паметта (RAM). Всяка инструкция заема 16 бита от паметта. Младшите 4 бита (b0-b3) са операционния код, от който се разбира каква точно е инструкцията. Следователно, този CPU не може да има повече от 16 инструкции ( $2^4 = 16$ ). Старшите 12 бита (b4-b15) са адреса на клетката, чието съдържание трябва да се прочете и зареди в акумулатора. Следователно, този CPU може да адресира максимално 4KB RAM ( $2^{12} = 4096$ ). Във всеки момент от време адресът на инструкцията, която следва да се изпълни, се сочи от програмния брояч *pc*. Съдържанието на тази клетка се зарежда в регистъра на инструкциите *ir*.

2. Декодиране на инструкцията. Анализират се младшите 4 бита от *ir* (операционния код), за да се разбере каква точно е инструкцията. В конкретният случай стойността на операционния код е 1, което съответства на инструкция *load*.

3. Изпълнение на инструкцията. След като се знае коя е инструкцията, следва нейното изпълнение. То се реализира на ниво микрокод. Микрокодът е програмният код,

който CPU може да изпълнява. В конкретният случай микрокодът, който изпълнява инструкцията *load*, е показан на Фиг. 3.



```
Execute sequence
ir(4-15)->mar
Main[mar]->mdr
mdr->acc
End
```

Фиг. 3. Микрокод, който реализира инструкцията *load*

Започва се с извличане на адреса на клетката от паметта, чието съдържание трябва да се зареди в акумулатора. Този адрес (битове 4-15 от регистър *ir*) се зарежда в регистър *mar* (Memory Address Register). Следва прочитане на съдържанието на клетката и зареждането му в регистър *mdr* (Memory Data Register). Съдържанието на *mdr* се прехвърля в акумулатора (*acc*) с което изпълнението на инструкцията *load* на ниво микрокод завършва с *End*.

### III. Задачи за изпълнение

#### Задача 1:

Анализирайте изпълнението на инструкции *store*, *jump* и *stop* на ниво микрокод.

#### Задача 2:

Създайте нова инструкция с име *inca*. Нейното предназначение е да увеличи с 1 текущото съдържание на акумулатора.

Помощ: Трябва да създадете нов микрокод, който инкрементира съдържанието на акумулатора. За целта изберете „Microinstructions” от меню „Modify”. От падащото меню „Type of Microinstruction” изберете „Increment”. Натиснете бутон „New”. Попълнете следната информация:

- *Name:* Inc-acc
- *Register:* acc
- *overflowBit:* none
- *carryBit:* none
- *Delta:* 1

Сега стартирайте „Machine Instructions” от меню „Modify”. Натиснете бутон „New” и въведете името на новата инструкция - *inca*. От поле „Format” задайте 4 бита за операционен код (по подразбиране 0xC) и оставете останалите битове празни (unused). От поле „Implementation” трябва да опишете микрокода, който реализира новата инструкция. За целта от „Microinstructions” изберете „Increment”, а след това Inc-acc. С влачене прехвърлете този микрокод в полето „Execute sequence”. Аналогично изберете микрокод „End” от папка „end”.

Тествайте новата инструкция чрез следния програмен код:

```

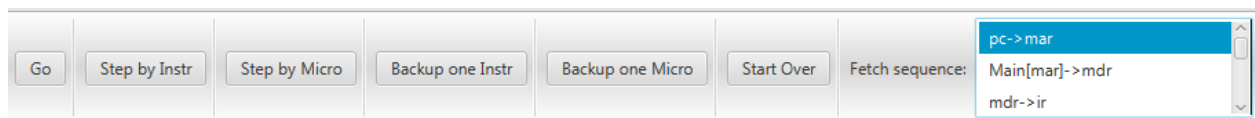
1 read
2 inca
3 write

```

Инструкция *read* чака въвеждане на число от конзолата и го записва в акумулатора. Инструкция *write* визуализира на конзолата съдържанието на акумулатора.

### Задача 3:

Тествайте програмният код от Задача 2 в режим *Debug*. За целта от меню „Execute” -> „Options” -> „Breakpoints” задайте за „Program counter” регистър *pc*. След това изпълнете „Execute” -> „Reset Everything”. Асемблирайте и заредете в паметта програмния код – „Execute” -> „Assemble & load”. Активирайте режим *Debug*: „Execute” -> „Debug mode”. Използвайте менюто на този режим (виж Фиг. 4).



Фиг. 4. Меню в режим *Debug*

Имате възможност да изпълните програмния код по различни начини:

- *Go* - без прекъсване след изпълнение на всяка инструкция,
- *Step by Instr* – изпълнява текущата инструкция. Може да разгледате как са се променили регистрите и паметта след изпълнение на инструкцията.
- *Step by Micro* – изпълнение на инструкциите на ниво микрокод (fetch sequence).
- *Backup one Instr* – анулира действието на последни изпълнената индструкция.
- *Backup one Micro* – анулира действието на последния микрокод.
- *Start Over* – изпълнението на кода започва отначало.

### Задача 4:

Като използвате режим *Debug* анализирайте микрокода (ред по ред), който изпълнява инструкция *read*.

## IV. Допълнителни въпроси

1. Защо производителността на един CPU зависи основно от неговата архитектура?
2. Систематизирайте процесорите на Intel (от 80486 до Core i9) в зависимост от вътрешната им архитектура.
3. Какво реализира следната последователност от микрокод:

```

pc -> mar
Main[mar] -> mdr
mdr -> ir
Inc2 - pc
decode - ir

```