Основни програмни модели за работа с процесорни инструкции

I. Въведение

Основната цел на настоящото упражнение е в неговия край Вие да можете да:

- Създавате основни процесорни инструкции с помощта на CPU симулатора.
- Изпълняване основни процесорни инструкции чрез симулатора.
- Използвате процесорни инструкции, с които да прехвърляте данни в регистрите и сравнявате стойностите, записани там, въвеждате и извличате данни от хардуерния стек, правите преходи към точно определени адреси от паметта, както и да събирате стойности, записани в регистрите.
- Обясните функциите на специалните процесорни регистри PC, SR и SP.

II. Процесорни симулатори

Използването на симулатори спомага за по-доброто разбиране на теоретичните концепции, които се описват по време на лекциите. Симулаторите осигуряват визуално и анимирано представяне на механизмите на работа на системите и дават възможност на обучаващите се да наблюдават отвътре самата работа на тези системи, която освен че остава скрита за потребителите, е и трудно и ли дори невъзможно да се представи по друг начин. Освен това чрез използването на симулатори се позволява на обучаващите се да експериментират и изследват различните технологични аспекти на системите, без да се налага да инсталират и конфигурират реални системи.

III. Основни аспекти

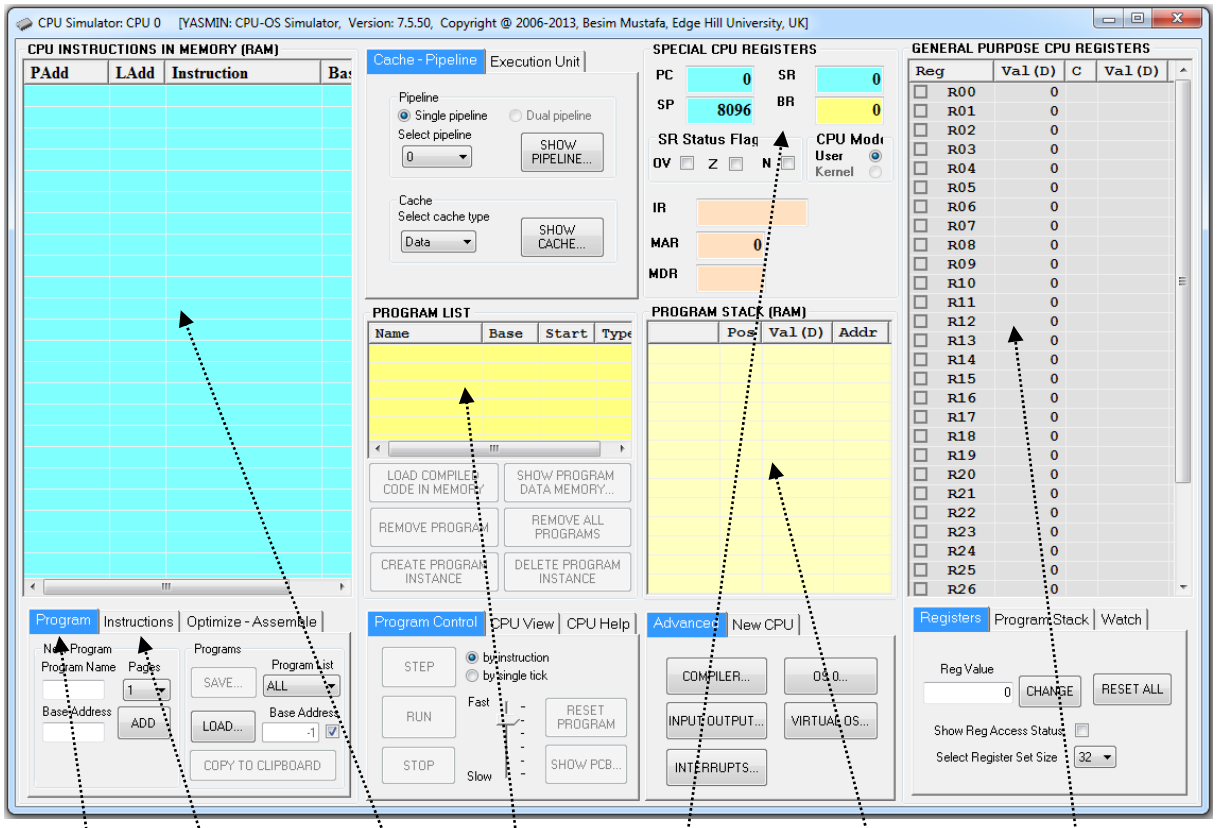
Програмният модел на компютърните архитектури дефинира архитектурните компоненти от ниско ниво, които участват пряко в работата на:

- Набора от инструкции на процесора
- Регистрите на процесора
- Различните видове адресиране на инструкциите и данните в тях

Програмният модел дефинира и взаимодействията между описаните по-горе компоненти. Всъщност именно програмния модел от ниско ниво е това, което позволява програмните изчисления.

IV. Описание на симулатора

Основният програмен прозорец на симулатора е съставен от няколко елемента, които представляват различни функционални компоненти в рамките на симулирания процесор.



Процесорни инструкции в паметта

Добавяне на инструкции в програмата

Създаване на програма

Списък на програмите

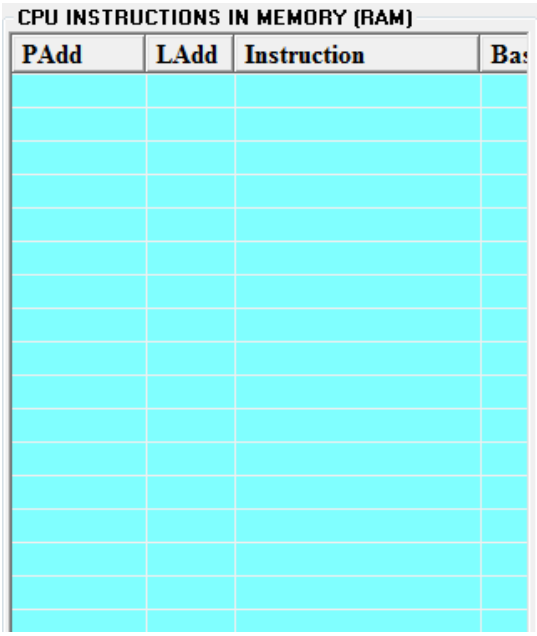
Специални процесорни регистри

Програмен стек

Процесорни регистри

Елементите от симулатора, които ще се използват по време на днешното упражнение, са описани по-долу. Моля прочете внимателно тяхното описание и ги намерете преди започване на същинската част от настоящото упражнение.

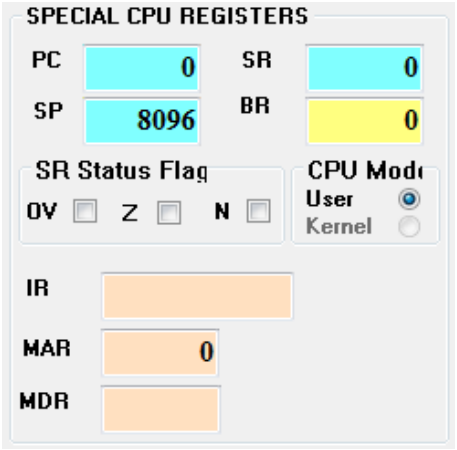
1. Процесорни инструкции в паметта



В този прозорец са показани инструкциите, от които се състои програмата. Те са представени като последователност от инструкционни мнемоники от ниско ниво (асемблерен вид), а не като бинарен код. По този начин кодът става по-ясен и по-лесно четим.

Всяка инструкция се асоциира с два адреса – физически (**PAdd**) и логически (**LAdd**). В този прозорец също се вижда и базовия адрес на всяка инструкция (**Base**). Всички инструкции от една програма имат един и същ базов адрес.

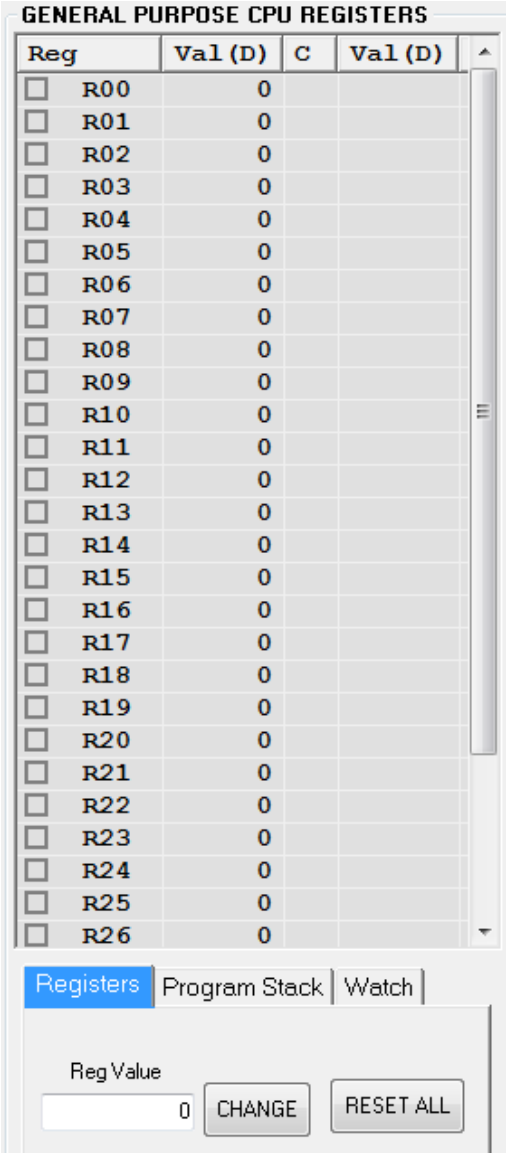
2. Специални процесорни регистри



В този прозорец са показани процесорните регистри, които имат предварително определени специализирани функции.
PC (Program Counter) – съдържа адреса на следващата инструкция, която трябва да се изпълни.
IR (Instruction Register) – съдържа инструкцията, която в момента се изпълнява.
SR (Status Register) – съдържа информация, която се отнася до резултата от последната изпълнена инструкция.
SP (Stack Pointer) – този регистър сочи към стойността, която се намира най-отгоре в програмния стек.
BR (Base Register) – съдържа текущия базов адрес.
MAR (Memory Address Register) – съдържа адреса от паметта, който в момента се достъпва.
MDR (Memory Data Register) – междинен регистър, който съдържа инструкцията, която в момента се изпълнява.
Status bits – битове за статус:

- **OV (Overflow)** – Препълване
- **Z (Zero)** – Нула
- **N (Negative)** – Отрицателен

3. Процесорни регистри

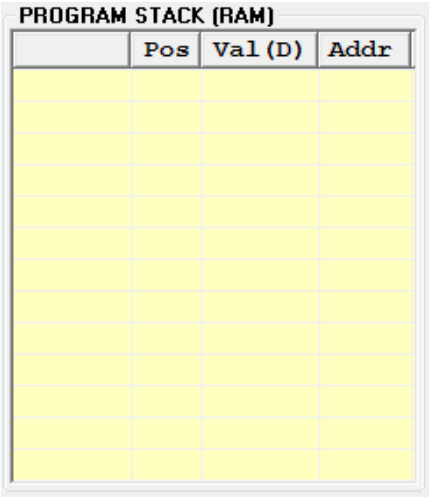


В този прозорец са показани стойностите на всички регистри с общо предназначение. Това са много бързи памети, които се използват за временно съхранение на данни по време на изпълнение на програмата. Най-често се използват за съхранение на променливи, използвани в програмните езици от високо ниво.

Броят на регистрите с общо предназначение варира в зависимост от архитектурата на процесора. Някои процесори имат повече такива регистри (напр. 128 регистъра), докато други по-малко (напр. 8 регистъра). Във всички случаи обаче те имат еднаква функция.

В този прозорец са показани имената на всички регистри с общо предназначение (**Reg**), техните текущи стойности (**Val**) и някои допълнителни елементи, които са резервирани за системата. Допълнително има възможност за ръчна промяна на стойността на всеки един от регистрите. Това става като най-напред се избере дадения регистър, след което в полето **Reg Value** се запише новата стойност и се натисне бутона **CHANGE**.

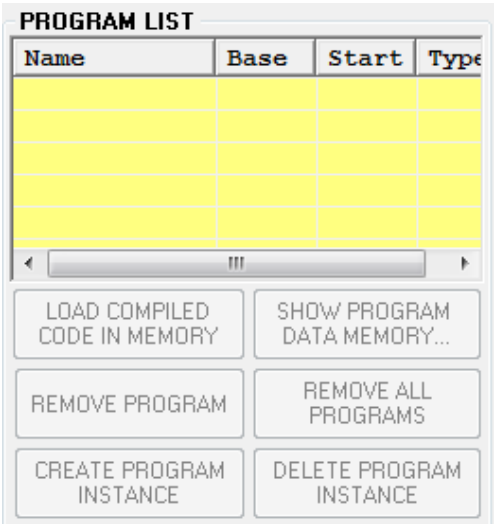
4. Програмен стек



Програмният стек е друго място, където временно се съхраняват данни по време на изпълнение на програмата. Структурата на стека е от тип LIFO (Last-In-First-Out). Той често се използва като средство за избягване на прекъсванията и извикване на подпрограми. Всяка програма има свой собствен стек.

Процесорните инструкции PSH (Push) и POP се използват за въвеждане и извличане на данни от върха на стека.

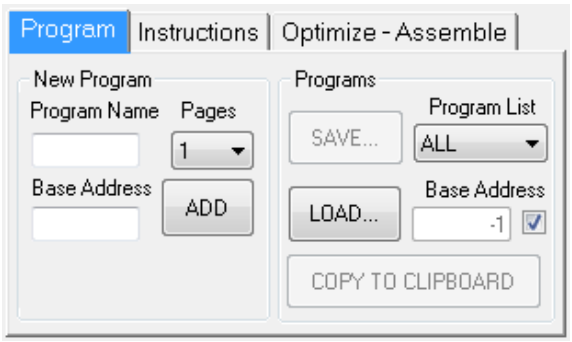
5. Списък на програмите



В този прозорец се визуализират създадените от нас програми.

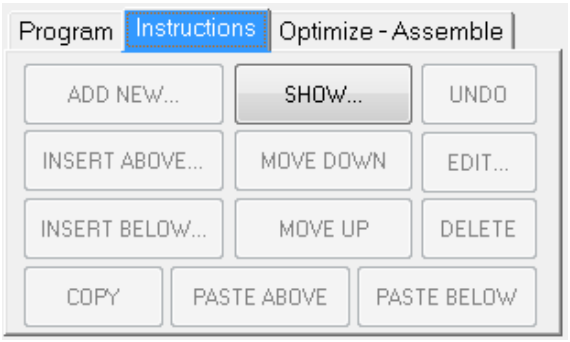
Бутонът **REMOVE PROGRAM** се използва за премахване на избрана програма от списъка, а чрез бутона **REMOVE ALL PROGRAMS** се премахват всички програми от списъка. Трябва да се има в предвид, че когато една програма се премахне от този списък, се премахват и всички нейни инструкции от Списъка с инструкциите.

6. Създаване на програма



За да се създаде нова програма в полето **Program Name** се изписва нейното име, а в **Base Address** нейния базовия адрес. Чрез бутона **ADD** се създава новата програма и нейното име се появява в Списъка с програмите.

7. Добавяне и редактиране на инструкции



ADD NEW… – добавяне на нова инструкция в програмата
EDIT… – редактиране на избраната инструкция
MOVE DOWN/MOVE UP – преместване на избраната инструкция надолу/нагоре
INSERT ABOVE…/INSERT BELOW… – добавяне на нова инструкция над/под избраната инструкция

V. Описание на лабораторното упражнение

За да може да създаваме и изпълняваме инструкции най-напред трябва да създадем програма. В табът **Program** първо въвеждаме **Име на програмата**, а след това и **Базов адрес** (това може да бъде всякакво число, но за това упражнение нека бъде 100). Натискаме бутона **ADD**. Новосъздадената програма с нейното име се появява в Списъка с програмите. Чрез бутона **SAVE…** се записва самата програма и нейните инструкции във файл, а чрез бутона **LOAD…** се зарежда записаната програма в симулатора.

След като сме изпълнили стъпките по-горе може да пристъпим към създаването на инструкции в симулатора. В таба **Instructions** натискаме бутона **ADD NEW…**, след което се появява нов прозорец **Instructions: CPU0**. От там избираме процесорната инструкция и евентуално настройваме нейните параметри, като за всяка инструкция има и кратко описание. В Приложение № 1 към настоящото упражнение има списък от няколко инструкции, както и примери за тяхното използване.

Вече сме готови да започнем същинската част от днешното упражнение. Отговорите на всяка от точките по-долу записвайте в определените за това текстови полета. *Препоръчително е редовно да записвате създадената програма във файл, така че ако симулатора поради някаква причина се повреди, да можете да го рестартирате и да заредите програмата от там, докъдето сте стигнали.*

1. Създайте инструкция, която премества (move) числото 5 в регистър R00.

2. Изпълнете горната инструкция като кликнете два пъти върху нея. Вижте какво се случва в Прозореца с процесорните инструкции.

3. Създайте инструкция, която премества (move) числото 8 в регистър R01.

4. Изпълнете я (като кликнете два пъти върху нея).

5. Вижте стойностите, записани в регистрите R00 и R01.

6. Създайте инструкция, която събира (add) съдържанието на R00 и R01.

7. Изпълнете я.

8. В кой регистър се записа резултата?

9. Създайте инструкция, която въвежда (push) горния резултат най-отгоре в хардуерния стек, след което я изпълнете.

10. Създайте инструкция, която въвежда (push) числото -2 най-отгоре в хардуерния стек и я изпълнете. Вижте какво се случва в Програмния стек.

11. Каква е стойността на регистъра SP? Когато въведете стойност в Програмния стек, стойността на регистъра SP се променя.

12. Създайте инструкция, която сравнява (compare) стойностите на регистрите R00 и R01.

13. Изпълнете я.

14. Каква е стойността на регистъра SR?

15. Какъв е статуса на флаговете OV/Z/N в Регистъра на състоянията (Status Register)? Кои полета са маркирани и кои не? Какво означава това?

16. Създайте инструкция, с която да направите безусловен преход (unconditional jump) към първата инструкция в програмата.

17. Изпълнете я.

18. Каква е стойността на регистъра PC? Не е ли това адреса на следващата инструкция за изпълнение? Към коя инструкция сочи той?

19. Погледнете стойностите, записани в колоните PAdd и LAdd. Какво означават те? Различни ли са?

20. Каква е разликата в стойностите на LAdd при първата и при втората инструкции? Какво означават тези стойности? Може ли да са свързани с дължината на инструкциите в байтове?

21. Създайте инструкция, която извлича (pop) стойността, записана най-отгоре в Програмния стек, и я записва в регистър R02.

22. Изпълнете я.

23. Каква е стойността на регистъра SP?

24. Създайте инструкция, която извлича (pop) стойността, записана най-отгоре в Програмния стек, и я записва в регистър R03.

25. Изпълнете я.

26. Каква е стойността на регистъра SP?

27. Изпълнете последната инструкция отново. Какво стана? Обяснете.

28. Създайте инструкция, която сравнява (compare) стойностите на регистрите R04 и R05.

29. Въведете ръчно две еднакви стойности за регистрите R04 и R05.

30. Още веднъж изпълнете сравнението от Точка 28.

31. Кой от флаговете OV/Z/N е установен (маркиран)? Защо?

32. Въведете ръчно стойност за регистър R05, която е по-голяма от тази на регистър R04.

33. Изпълнете сравнението от Точка 28.

34. Кой от флаговете OV/Z/N е установен? Защо?

35. Въведете ръчно стойност за регистър R04, която е по-голяма от тази на регистър R05.

36. Изпълнете сравнението от Точка 28.

37. Кой от флаговете OV/Z/N е установен? Защо?

38. Създайте инструкция, с която да направите безусловен преход към първата инструкция в програмата, ако стойностите на регистрите R04 и R05 са еднакви.

39. Тествайте горна инструкция като въведете ръчно еднакви стойности на регистрите R04 и R05. Когато сте готови първо изпълнете инструкцията за сравнение и едва след това самата инструкция от Точка 38. Получи ли се?

40. Съхранете създадената програма чрез бутона **SAVE…**

41. Създадената програма представете под формата на протокол.

Приложение № 1

Инструкция	Описание
Инструкции за трансфер на данни	
MOV	Премества данни в регистър Премества данни от един регистър в друг регистър **MOV #2, R01** – Премества числото 2 в регистър R01 **MOV R01, R03** – Премества съдържанието на регистър R01 в R03
LDB	Зарежда байт от паметта в регистър
LDW	Зарежда дума (2 байта) от паметта в регистър
STB	Съхранява байт от регистър в паметта
STW	Съхранява дума (2 байта) от регистър в паметта
PSH	Въвежда данни най-отгоре в хардуерния стек Въвежда данни от регистър най-отгоре в хардуерния стек **PSH #6** – Въвежда числото 6 най-отгоре в стека **PSH R03** – Въвежда съдържанието на регистър R03 най-отгоре в стека
POP	Извлича данните, които се намират най-отгоре в хардуерния стек, и ги записва в регистър **POP R05** – Извлича данните, които се намират най-отгоре в стека, и ги записва в регистър R05 *Ако се опитаме да извлечем данни от празен стек се получава грешка „Препълване на стека (Stack overflow)“*
Аритметични инструкции	
ADD	Събира число с регистър Събира регистър с регистър **ADD #3, R02** – Събира числото 3 със съдържанието на регистър R02 и записва резултата в R02 **ADD R00, R01** – Събира съдържанието на регистър R00 със съдържанието на регистър R01 и записва резултата в R01
SUB	Изважда число от регистър Изважда регистър от регистър
MUL	Умножава число с регистър Умножава регистър с регистър
DIV	Разделя число с регистър Разделя регистър с регистър

Инструкции за контрол на трансфера	
JMP	Безусловен преход към адрес на инструкция **JMP 100** – Безусловен преход към адрес 100
JLT	Преход към адрес на инструкция, ако е по-малко от (след последното сравнение)
JGT	Преход към адрес на инструкция, ако е по-голямо от (след последното сравнение)
JEQ	Преход към адрес на инструкция, ако е равно (след последното сравнение) **JEQ 200** – Преход към адрес 200, ако при предишното сравнение са сравнени две еднакви числа, т.е. флага Z е установен
JNE	Преход към адрес на инструкция, ако не е равно (след последното сравнение)
CAL	Преход към адрес на подпрограма
RET	Връщане от подпрограма
SWI	Софтуерно прекъсване
HLT	Стоп на симулация
Инструкции за сравнение	
CMP	Сравнява число с регистър Сравнява регистър с регистър **CMP #5, R02** – Сравнява числото 5 със съдържанието на регистър R02 **CMP R01, R03** – Сравнява съдържанието на регистрите R01 и R03 Ако R01 = R03, тогава флага Z се установява Ако R01 < R03, тогава никои от флаговете не се установява Ако R01 > R03, тогава флага N се установява
Инструкции за вход и изход	
IN	Извлича входни данни (ако са налични) от външно входно-изходно устройство
OUT	Изпраща данни към външно входно-изходно устройство