

Машинен език и програмиране на асемблер

Лабораторно упражнение №1

Цел

- Как се пишат и асемблират програми за MIPS архитектура?
- Как се изпълняват инструкциите?
- За какво служи регистърът Program Counter (PC)?
- Къде са разположени данните и инструкциите по време на изпълнение на програмата?

Инструменти

- Необходими инструменти за това упражнение:

- MipsIt.exe



- за редактиране на програмни код

- Mips.exe



- за симулация

План

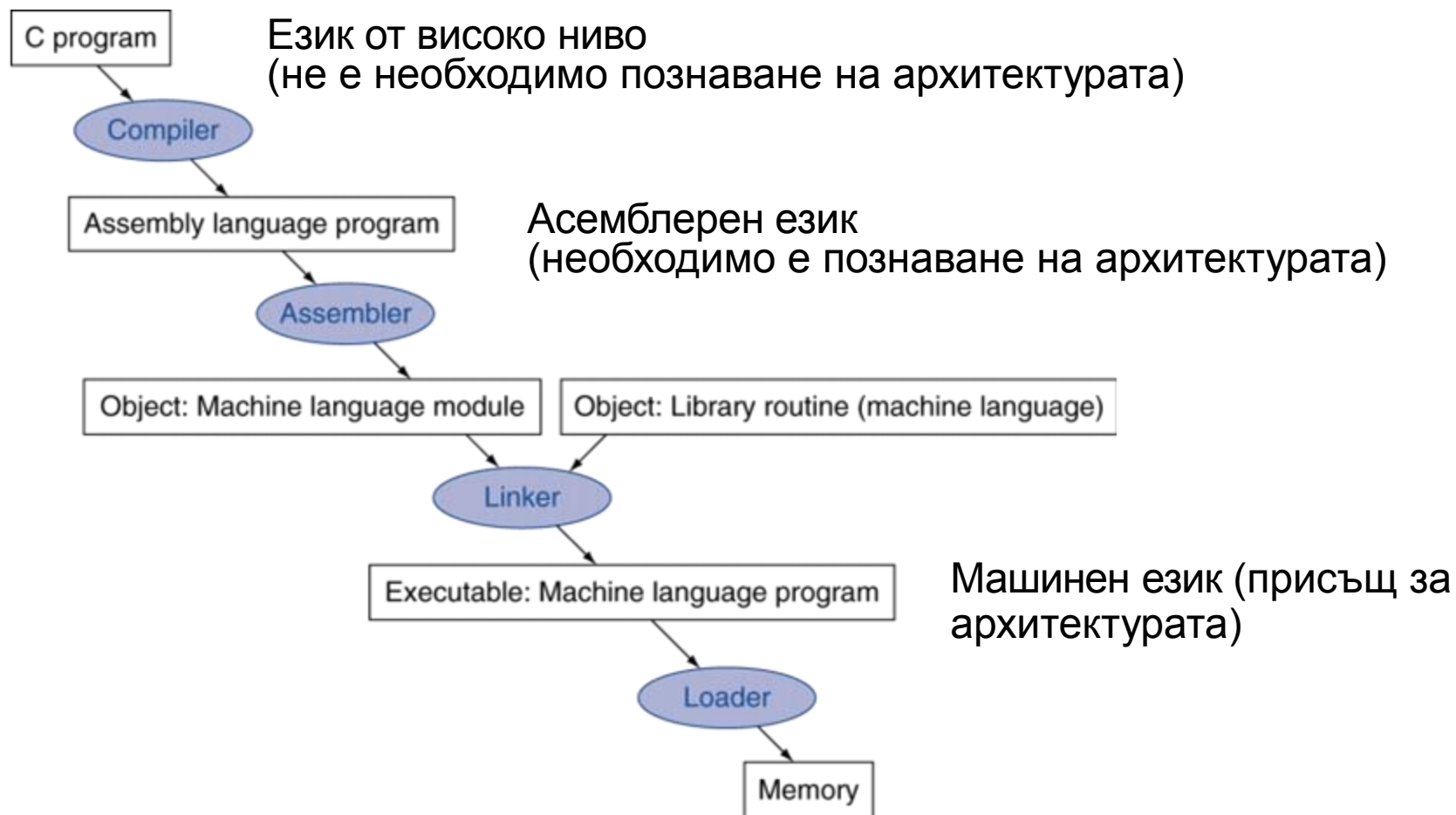
- Теоретична част
- MIPS архитектура
- Писане на програми на асемблер за MIPS архитектура
- Писане на програми на асемблер с инструментите на MipsIT

Програмисти vs. компютри

- Програмистите могат да пишат програми на език от високо ниво или на асемблерен език



- Компютрите могат само да изпълняват програми написани на собственият им език (машинен код)



Машинен език

- CPU може единствено да изпълнява машинни инструкции
- Инструкциите са разположени в паметта заедно с данните
- Машинната инструкция е последователност от битове

00001	101010111101110	011
-------	-----------------	-----

Опкод

Операнд (памет)

Операнд (регистър)

- Съществува набор от машинни инструкции, които се поддържат от дадена архитектура (Instruction Set)

Instruction Set

- Тип на инструкциите
 - Аритметично-логически (ALU)
 - Трансфер на данни
 - Преходи
 - В/И
- Определя:
 - Типа на операндите и операциите
 - Къде се намират операндите (в регистрите или паметта)
 - Режима за адресиране (директно, непосредствено, косвено и др.)
 - Формата на инструкцията (фиксирана дължина или променлива дължина)

MIPS архитектура

- Регистров файл, който се състои от 32 32-битови регистри с общо предназначение (видими за програмиста)
- Регистрите са означени като \$0 - \$31
- Регистърът \$ 0 = 0, е константа (не може да бъде модифицирана от програмиста)
- Регистърът \$ 31 се използва за съхраняване на адреса за връщане от подпрограма
- Специалните регистри Lo и Hi, които не са директно достъпни за програмиста, се използват за съхраняване на резултат от умножение или деление

MIPS архитектура

- Всички инструкции имат една и съща дължина (32-bits)
- Три формата за инструкции:

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

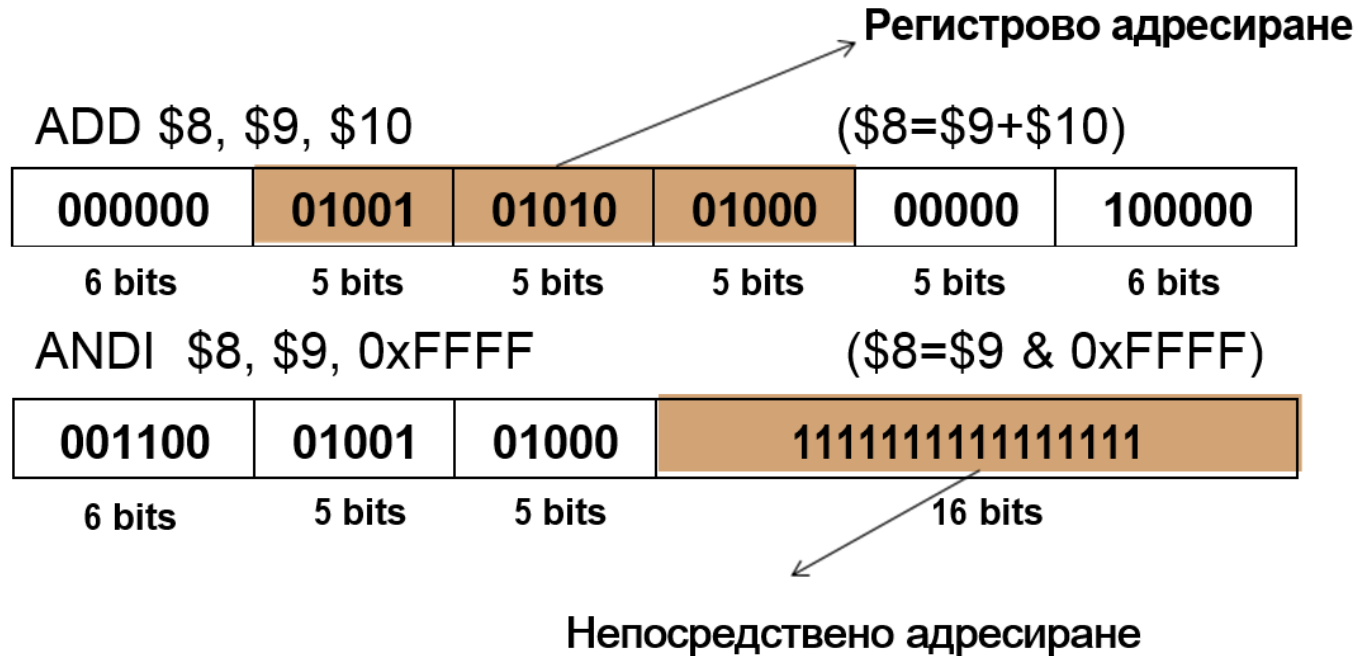
op	address
6 bits	26 bits

MIPS архитектура

- Регистрово адресиране:
 - Всички операнди са регистри
- Непосредствено адресиране:
 - Операндът е вграден в инструкцията
- Адресиране с отместване:
 - Адресът на паметта е посочен като отместване от регистър
- Относително адресиране, спрямо РС:
 - Адресът на паметта е посочен като отместване спрямо програмния брояч (РС)
- Псевдодиректно адресиране:
 - Адресът на паметта е (най-често) вграден в инструкцията
- Регистрово пряко адресиране:
 - Адресът на паметта се съхранява в регистър

MIPS архитектура

- Пример:

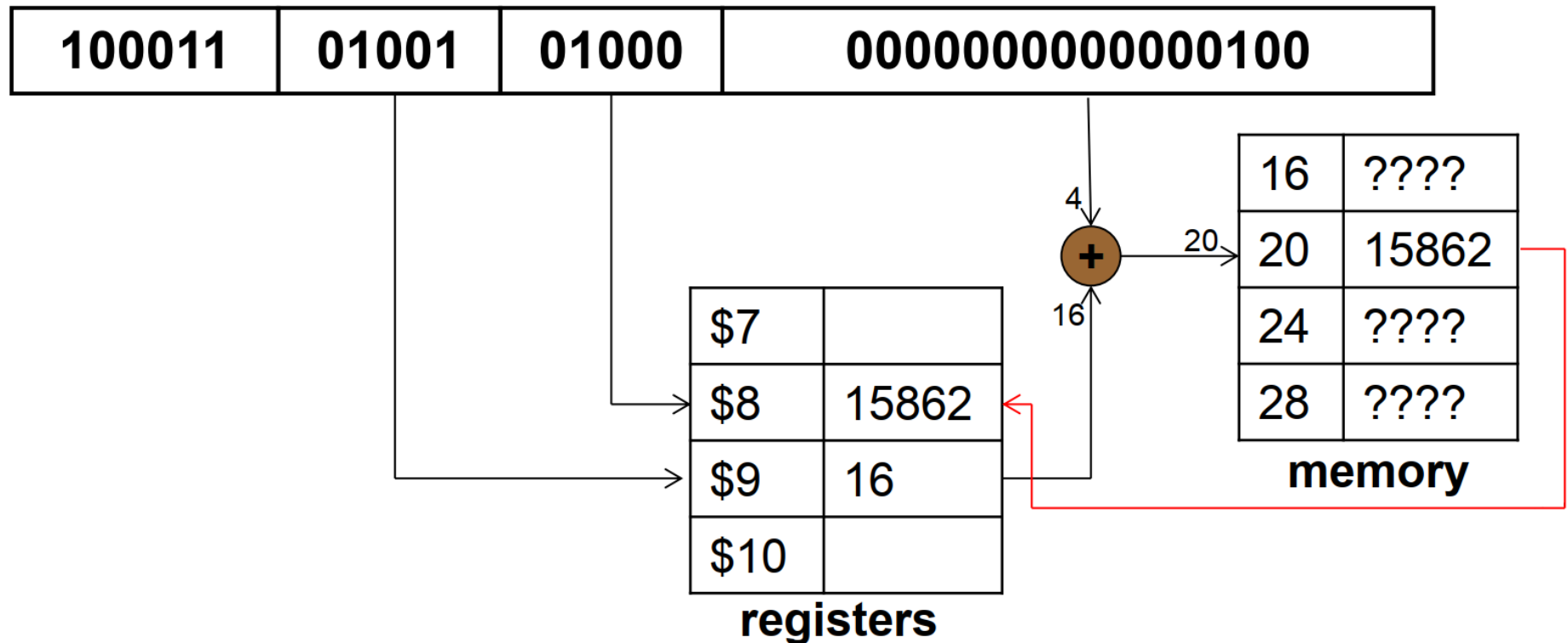


MIPS архитектура

- Пример:

LW \$8, 4(\$9)

(\$8=mm(\$9+4))



Програмиране на асемблер

- Инструкция на асемблер (мнемоничен израз) за всяка машинна инструкция

ADD \$8, \$9, \$10 == 0x12A4020

- Псевдоинструкции (поддържат се от програмата за асемблиране)
 - Превеждат се в последователност от машинни инструкции

LW \$8, data == LI \$9, 0x85000
 LW \$8, 0(\$9)

Програмиране на асемблер

- Структура на програмата
 - Обикновен текстов файл с декларации за данни и програмен код
 - Декларации за данни
 - Започват с директивата **.data**
 - Var_name: тип стойност(и)
 - Програмен код
 - Започва с директива **.text**
 - Съдържа инструкции
 - Определя началната и крайната точка на програмата
 - Допълнителни директиви – могат да се използват преди сегмента за деклариране на данни

Пример 1

The image shows a screenshot of an assembly editor interface. On the left, the source code is displayed, featuring comments and assembly directives. On the right, a 'Memory' window shows a table of memory addresses, their contents, and the corresponding assembly instructions. A diagram in the top right corner illustrates the system architecture, showing a CPU connected to a D-Cache.

Source Code:

```
#LINES STARTING WITH THIS SYMBOL ARE COMMENTS, AND ARE IGNORED
#INSTRUCTS THE ASSEMBLER TO PRESERVE THE ORDER OF THE INSTRUCTIONS
.set noreorder

#ASSEMBLY DIRECTIVE TO NOTIFY THE ASSEMBLER THAT THE FOLLOWING
.data
N: .word 0x0ff
Y: .word

#ASSEMBLY DIRECTIVE TO NOTIFY THE ASSEMBLER THAT THE FOLLOWING
.text

#ASSEMBLY DIRECTIVE TO DENOTE A GLOBAL SYMBOL
.globl start

#ASSEMBLY DIRECTIVE TO DENOTE A LOCAL SYMBOL
.ent start

start: lw $t2, N
      li $s, 0
      li $9, 0
      li $t0, 1
      li $t1, 1
again: addi $t1, $t1, 1
      add $s, $s, $0
      add $9, $t0, $0
      add $t0, $s, $9
      bne $t1, $t2, again
      nop
done:  sw $t0, Y
      nop

#ASSEMBLY DIRECTIVE TO DENOTE A LOCAL SYMBOL
.end start
```

Memory Window:

Address	Content	Label	Instruction
8001FFE0	00 00 00 00		NOP
8001FFE4	00 00 00 00		NOP
8001FFE8	00 00 00 00		NOP
8001FFEC	00 00 00 00		NOP
8001FFF0	00 00 00 00		NOP
8001FFF4	00 00 00 00		NOP
8001FFF8	00 00 00 00		NOP
8001FFFC	00 00 00 00		NOP
80020000	3C 0C 80 02	start()	LUI \$t2, 0x8002
80020004	8D 8C 00 40		LW \$t2, 0x40(\$t2)
80020008	24 08 00 00		ADDIU \$08, \$00, 0x0
8002000C	24 09 00 00		ADDIU \$09, \$00, 0x0
80020010	24 0A 00 01		ADDIU \$t0, \$00, 0x1
80020014	24 0B 00 01		ADDIU \$t1, \$00, 0x1
80020018	21 6B 00 01	again:	ADDI \$t1, \$t1, 0x1
8002001C	01 20 40 20		ADD \$08, \$09, \$00
80020020	01 40 48 20		ADD \$09, \$t0, \$00
80020024	01 09 50 20		ADD \$t0, \$08, \$09
80020028	15 6C FF FB		BNE \$t2, \$t1, 0xffffb
8002002C	00 00 00 00		NOP
80020030	3C 01 80 02	done:	LUI \$01, 0x8002
80020034	AC 2A 00 44		SW \$t0, 0x44(\$01)
80020038	00 00 00 00		NOP
8002003C	00 00 00 00		NOP
80020040	00 00 00 07	N:	SRAV \$00, \$00, \$00
80020044	00 00 00 0D	Y:	BREAK
80020048	00 00 00 00		NOP

Diagram: A block labeled 'CPU' is connected to a block labeled 'D-Cache'.

Пример 2

The screenshot displays an assembly editor with MIPS code on the left and a memory dump on the right. A red box highlights the code, and a green box highlights the memory dump entries for K, N, and Y.

Assembly Code:

```
K: .word 2
N: .word 0x0FD55
Y: .word

#ASSEMBLY DIRECTIVE TO NOTIFY THE ASSEMBLER THAT THE FOLLOWING IS TEXT
.text

#ASSEMBLY DIRECTIVE TO DENOTE A GLOBAL SYMBOL
.globl start

#ASSEMBLY DIRECTIVE TO DENOTE THE START OF THE PROGRAM
.ent start
.start:
    lw $9, K
    li $10, 32
    sub $10, $10, $9
    li $8, 0
    li $12, 0
check:
    nop
    beq $9, $0, proceed
    nop
    sll $8, $8, 1
    addi $8, $8, 1
    addi $9, $9, -1
    j check
proceed:
    lw $9, N
checka:
    and $11, $8, $9
    bne $11, $8, skip
    nop
    addi $12, $12, 1
skip:
    beq $10, $0, done
    nop
    sll $8, $8, 1
    addi $10, $10, -1
    j checka
done:
    sw $12, Y
    nop

#ASSEMBLY DIRECTIVE TO DENOTE THE END OF THE PROGRAM
.end start
```

Memory Dump:

Address	Content	Label
8001FFFF	00 00 00 00	NOP
80020000	3C 09 80 02	start: LUI \$09, 0x8002
80020004	8D 29 00 80	LW \$09, 0x80(\$09)
80020008	24 0A 00 20	ADDIU \$10, \$00, 0x20
8002000C	01 49 50 22	SUB \$10, \$10, \$09
80020010	24 08 00 00	ADDIU \$08, \$00, 0x0
80020014	24 0C 00 00	ADDIU \$12, \$00, 0x0
80020018	00 00 00 00	check: NOP
8002001C	11 20 00 06	BEQ \$00, \$09, 0x6
80020020	00 00 00 00	NOP
80020024	00 08 40 40	SLL \$08, \$08, 1
80020028	21 08 00 01	ADDI \$08, \$08, 0x1
8002002C	21 29 FF FF	ADDI \$09, \$09, 0xffff
80020030	08 00 80 06	J 0x8006
80020034	00 00 00 00	NOP
80020038	3C 09 80 02	proceed: LUI \$09, 0x8002
8002003C	8D 29 00 84	LW \$09, 0x84(\$09)
80020040	01 09 58 24	checka: AND \$11, \$08, \$09
80020044	15 68 00 02	BNE \$08, \$11, 0x2
80020048	00 00 00 00	NOP
8002004C	21 8C 00 01	ADDI \$12, \$12, 0x1
80020050	00 00 00 00	skip: NOP
80020054	11 40 00 05	BEQ \$00, \$10, 0x5
80020058	00 00 00 00	NOP
8002005C	00 08 40 40	SLL \$08, \$08, 1
80020060	21 4A FF FF	ADDI \$10, \$10, 0xffff
80020064	08 00 80 10	J 0x8010
80020068	00 00 00 00	NOP
8002006C	3C 01 80 02	done: LUI \$01, 0x8002
80020070	AC 2C 00 88	SW \$12, 0x88(\$01)
80020074	00 00 00 00	NOP
80020078	00 00 00 00	NOP
8002007C	00 00 00 00	NOP
80020080	00 00 00 02	K: SRL \$00, \$00, 0
80020084	00 00 FD 55	N: ???
80020088	00 00 00 05	Y: ???
8002008C	00 00 00 00	NOP
80020090	00 00 00 00	NOP

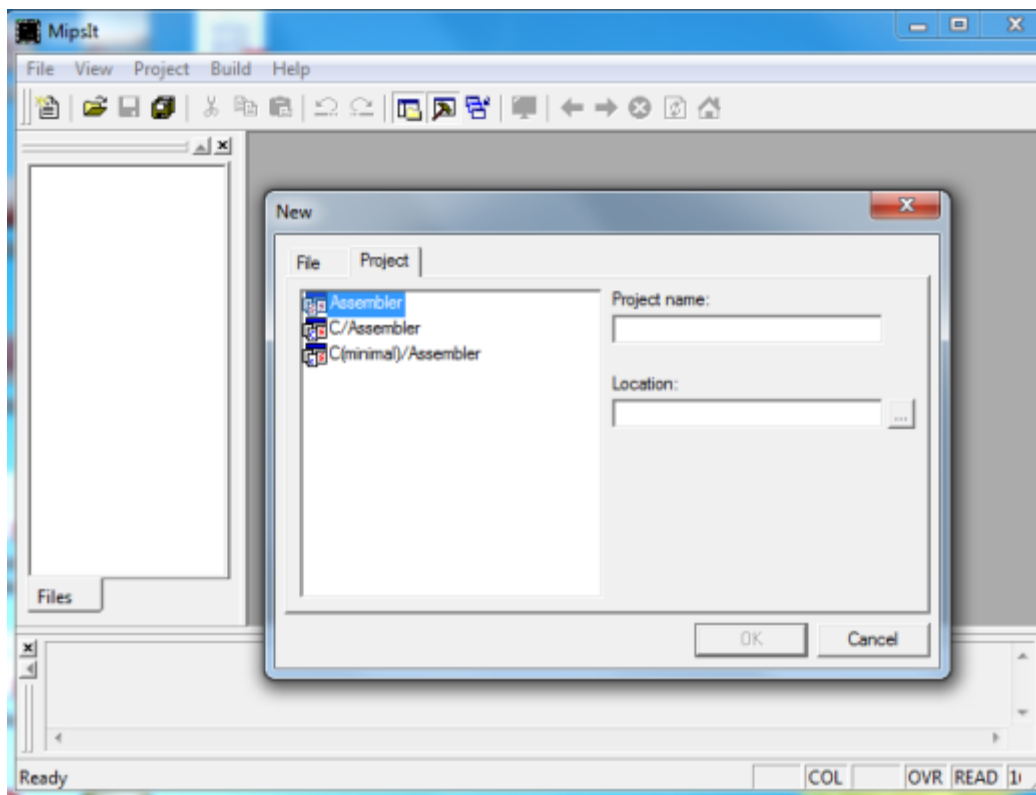
Програмиране на асемблер с използване на инструментите MipsIt

- Отворете MipsIt.exe



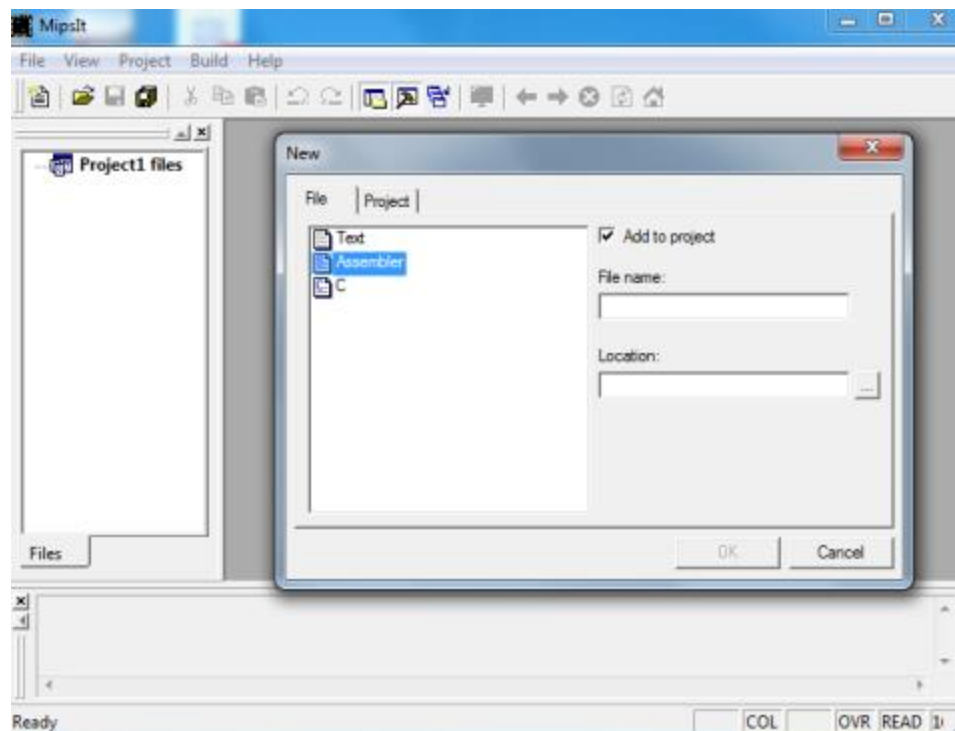
Програмиране на асемблер с използване на инструментите MipsIt

- Създайте нов проект на асемблер



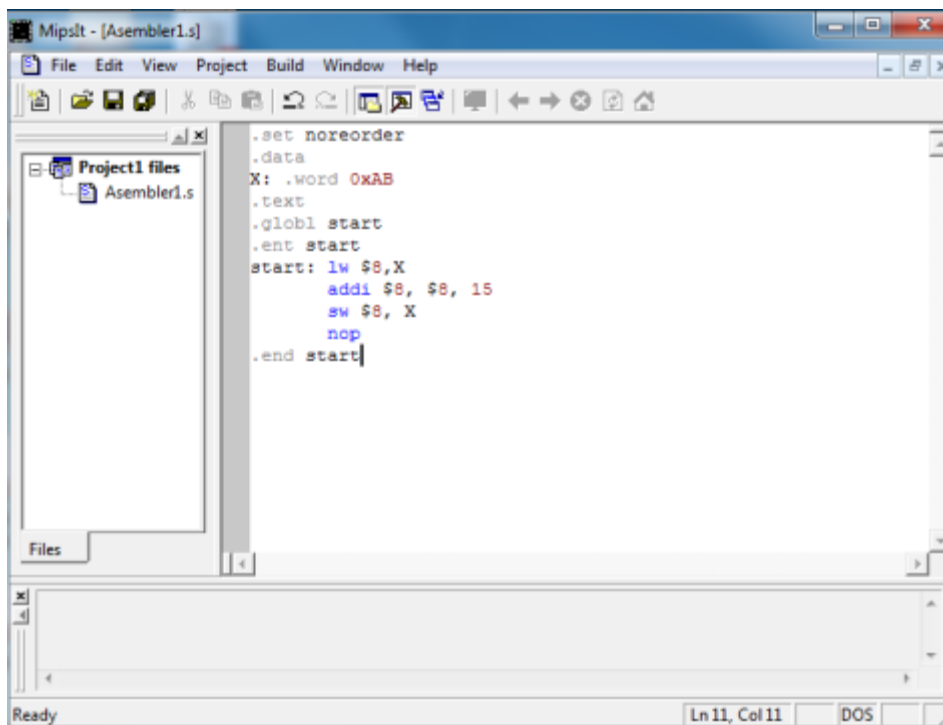
Програмиране на асемблер с използване на инструментите MipsIt

- Създайте нов асемблер файл



Програмиране на асемблер с използване на инструментите MipsIt

- Напишете програмата на асемблер



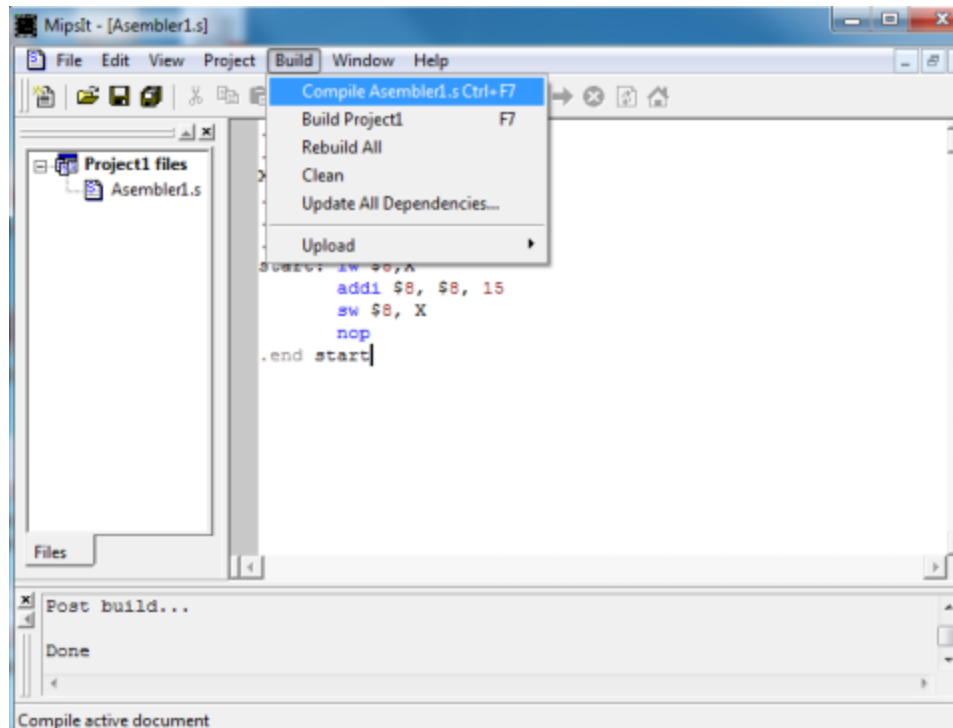
The screenshot shows a window titled "MipsIt - [Assembler1.s]" with a menu bar (File, Edit, View, Project, Build, Window, Help) and a toolbar. On the left is a "Project1 files" tree showing "Assembler1.s". The main text area contains the following assembly code:

```
.set noreorder
.data
X: .word 0xAB
.text
.globl start
.ent start
start: lw $8, X
      addi $8, $8, 15
      sw $8, X
      nop
.end start
```

At the bottom, a status bar shows "Ready", "Ln 11, Col 11", and "DOS".

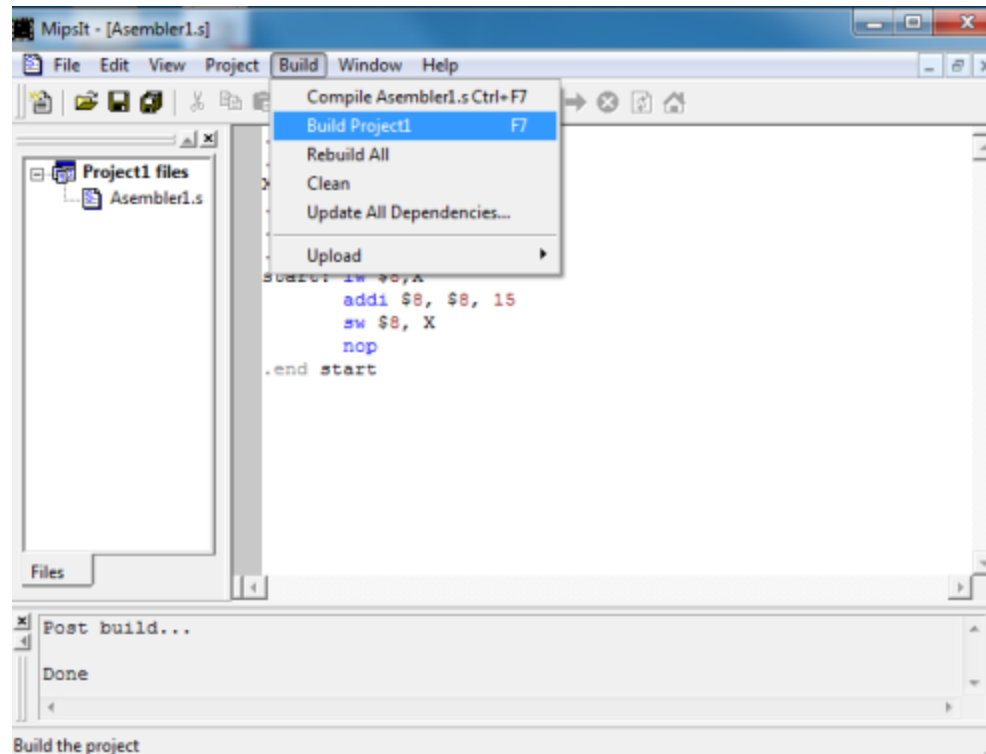
Програмиране на асемблер с използване на инструментите MipsIt

- Компилирайте и асемблирайте програмата



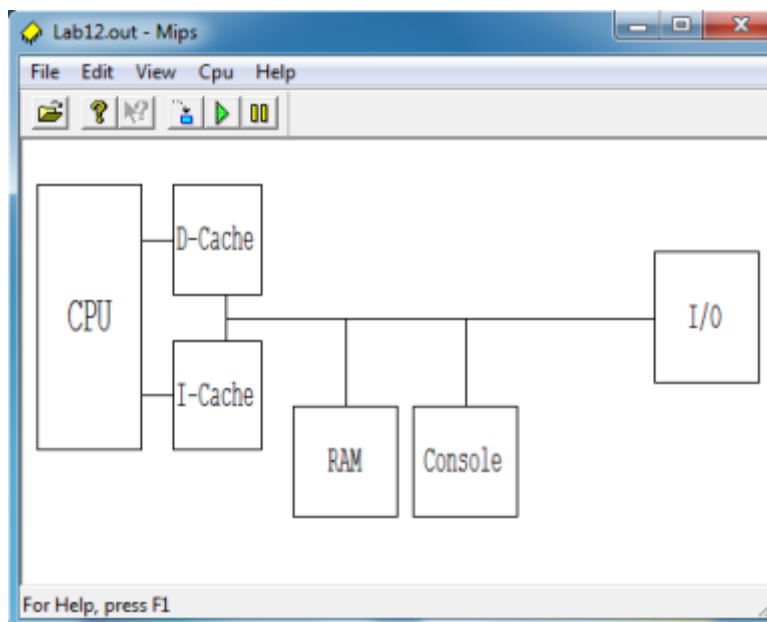
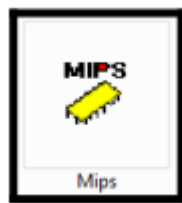
Програмиране на асемблер с използване на инструментите MipsIt

- Генерирайте изпълнимия код (Build the project)



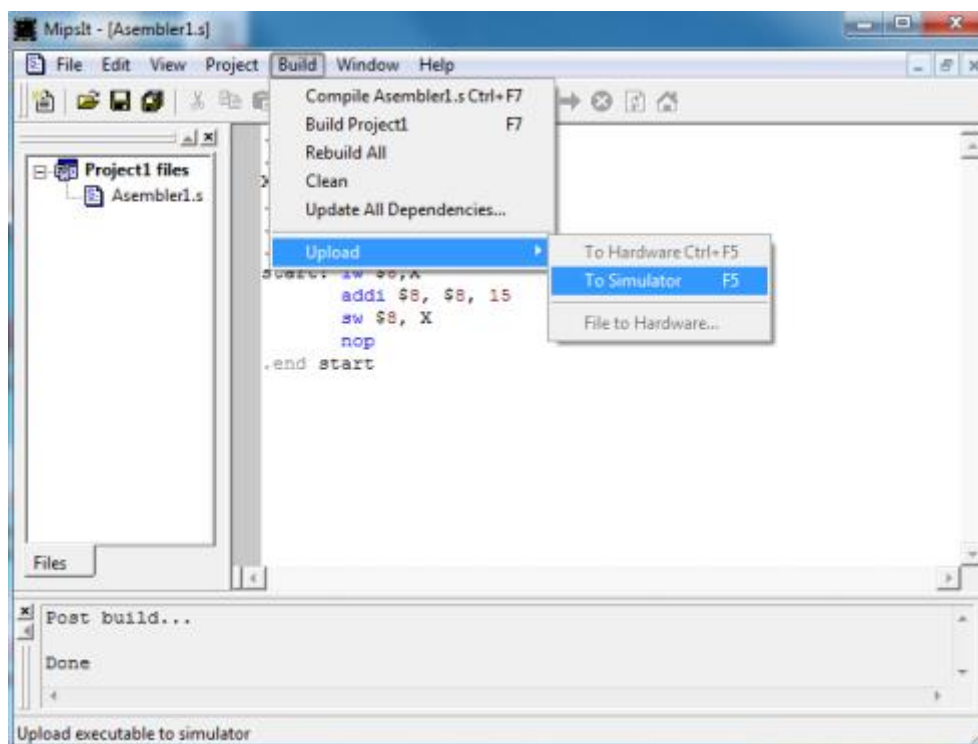
Програмиране на асемблер с използване на инструментите MipsIt

- Отворете Mips.exe



Програмиране на асемблер с използване на инструментите MipsIt

- Заредете програмата в симулатора



Програмиране на асемблер с използване на инструментите MipsIt



Програмиране на асемблер с използване на инструментите MipsIt

Address	Content	Label
8001FFC8	00 00 00 00	NOP
8001FFCC	00 00 00 00	NOP
8001FFD0	00 00 00 00	NOP
8001FFD4	00 00 00 00	NOP
8001FFD8	00 00 00 00	NOP
8001FFDC	00 00 00 00	NOP
8001FFE0	00 00 00 00	NOP
8001FFE4	00 00 00 00	NOP
8001FFE8	00 00 00 00	NOP
8001FFEC	00 00 00 00	NOP
8001FFF0	00 00 00 00	NOP
8001FFF4	00 00 00 00	NOP
8001FFFB	00 00 00 00	NOP
8001FFFC	00 00 00 00	NOP
80020000	3C 08 80 02	start() LUI \$08, 0x8002
80020004	8D 08 00 20	LW \$08, 0x20(\$08)
80020008	21 08 00 0F	ADDI \$08, \$08, 0xF
8002000C	3C 01 80 02	LUI \$01, 0x8002
80020010	AC 28 00 20	SW \$08, 0x20(\$01)
80020014	00 00 00 00	NOP
80020018	00 00 00 00	NOP
8002001C	00 00 00 00	NOP
80020020	00 00 00 AB	??? X
80020024	00 00 00 00	NOP
80020028	00 00 00 00	NOP
8002002C	00 00 00 00	NOP
80020030	00 00 00 00	NOP

ADDIU \$26, \$26, 0x8428 ; \$26=A0020000

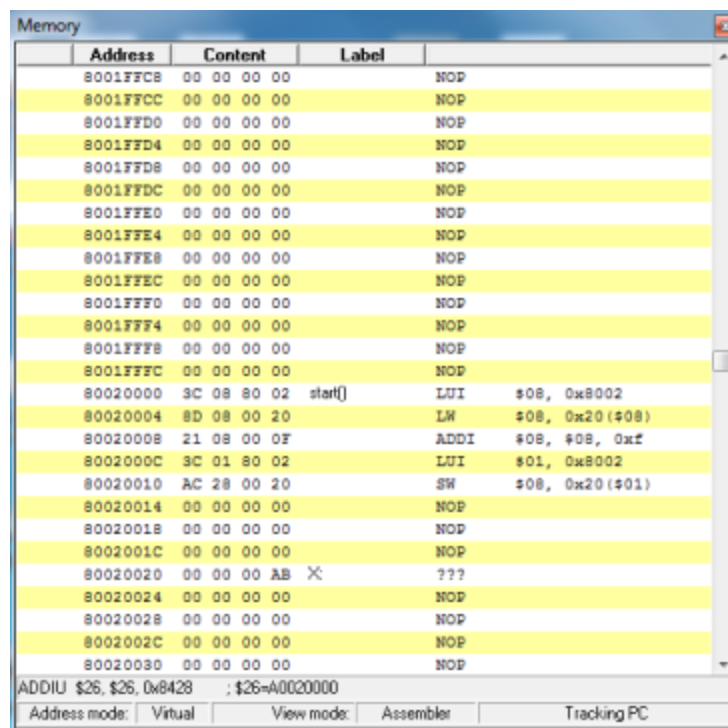
Address mode: Virtual View mode: Assembler Tracking PC

Сегмент с кода

Сегмент с данните

Програмиране на асемблер с използване на инструментите MipsIt

В прозореца Memory щракнете двукратно върху адреса от паметта, който е точно под последната инструкция в програмния код



The screenshot shows the 'Memory' window in the MipsIt tool. It displays a list of memory addresses and their corresponding assembly instructions. The instructions are mostly NOP (No Operation) instructions, followed by a sequence of instructions starting at address 80020000. The instructions include LUI, LW, ADDI, and SW, with various register and immediate values. The window also shows the current address mode as 'Virtual', the view mode as 'Assembler', and the tracking PC as '\$26-A0020000'.

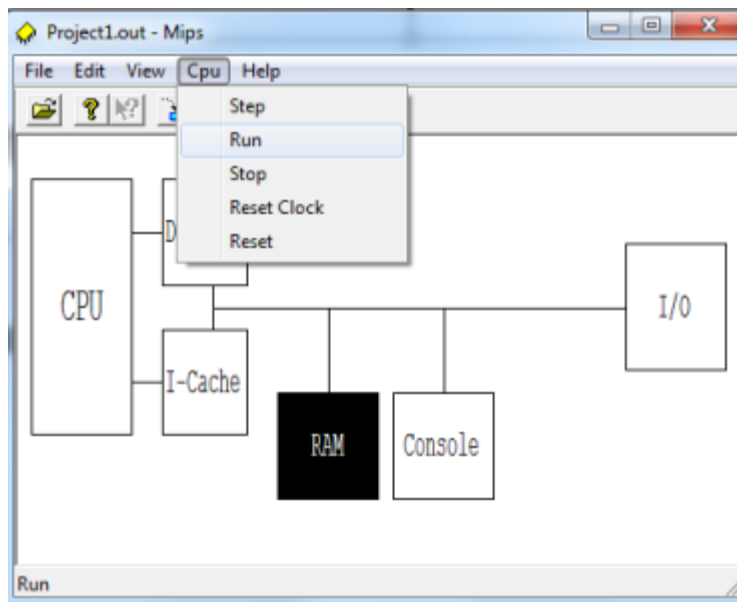
Address	Content	Label
8001FFC8	00 00 00 00	NOP
8001FFCC	00 00 00 00	NOP
8001FFD0	00 00 00 00	NOP
8001FFD4	00 00 00 00	NOP
8001FFD8	00 00 00 00	NOP
8001FFDC	00 00 00 00	NOP
8001FFE0	00 00 00 00	NOP
8001FFE4	00 00 00 00	NOP
8001FFE8	00 00 00 00	NOP
8001FFEC	00 00 00 00	NOP
8001FFF0	00 00 00 00	NOP
8001FFF4	00 00 00 00	NOP
8001FFF8	00 00 00 00	NOP
8001FFFC	00 00 00 00	NOP
80020000	3C 08 80 02	start() LUI \$08, 0x8002
80020004	8D 08 00 20	LW \$08, 0x20(\$08)
80020008	21 08 00 0F	ADDI \$08, \$08, 0xf
8002000C	3C 01 80 02	LUI \$01, 0x8002
80020010	AC 28 00 20	SW \$08, 0x20(\$01)
80020014	00 00 00 00	NOP
80020018	00 00 00 00	NOP
8002001C	00 00 00 00	NOP
80020020	00 00 00 AB X	???
80020024	00 00 00 00	NOP
80020028	00 00 00 00	NOP
8002002C	00 00 00 00	NOP
80020030	00 00 00 00	NOP

ADDIU \$26, \$26, 0x8428 ; \$26=A0020000

Address mode: Virtual View mode: Assembler Tracking PC

Програмиране на асемблер с използване на инструментите MipsIt

- Изпълнете програмата



The screenshot shows the 'Memory' window with a table of memory addresses, contents, and labels. The address 80020014 is highlighted in blue. A red vertical line is positioned at address 80020004.

Address	Content	Label
8001FFDC	00 00 00 00	NOP
8001FFE0	00 00 00 00	NOP
8001FFE4	00 00 00 00	NOP
8001FFE8	00 00 00 00	NOP
8001FFEC	00 00 00 00	NOP
8001FFF0	00 00 00 00	NOP
8001FFF4	00 00 00 00	NOP
8001FFF8	00 00 00 00	NOP
8001FFFC	00 00 00 00	NOP
80020000	3C 08 80 02	start() LUI \$08, 0x8002
80020004	8D 08 00 20	LW \$08, 0x20(\$08)
80020008	21 08 00 0F	ADDI \$08, \$08, 0xF
8002000C	3C 01 80 02	LUI \$01, 0x8002
80020010	AC 28 00 20	SW \$08, 0x20(\$01)
80020014	00 00 00 00	NOP
80020018	00 00 00 00	NOP
8002001C	00 00 00 00	NOP
80020020	00 00 00 AB	X: ???
80020024	00 00 00 00	NOP
80020028	00 00 00 00	NOP
8002002C	00 00 00 00	NOP
80020030	00 00 00 00	NOP
80020034	00 00 00 00	NOP
80020038	00 00 00 00	NOP
8002003C	00 00 00 00	NOP
80020040	00 00 00 00	NOP
80020044	00 00 00 00	NOP

NOP

Address mode: Virtual View mode: Assembler Tracking PC

- Може да изпълнявате постъпково като изберете Step

Програмиране на асемблер с използване на инструментите MipsIt

- За по-добро разбиране на последователността на изпълнение използвайте командата Step

