



Universiteit Antwerpen
| Faculteit Wetenschappen

Computersystemen en -architectuur

Introductie MIPS

Kasper Engelen

kasper.engelen@uantwerpen.be

Academiejaar 2022 – 2023

Instructies

- Bepalen wat een programma doet
- Worden één voor één uitgevoerd
- Arithmetic: optellen, delen, and, or, bitshifts, etc.
- Load/Store: lezen/schrijven naar geheugen
- Conditional: if-statements, loops, etc.
- Voorbeeld:
 - `li $t1, 12` # Register \$t1 bevat nu de waarde 12
 - `addi $t0, $t1, 2` # $\$t0 = \$t1 + 2$
 - Register \$t0 bevat nu de waarde 14

Registers

| Naam | Nummer | Betekenis |
|--------------|-----------------|-------------------------------|
| \$zero | 0 | Altijd gelijk aan 0 |
| \$at | 1 | Gereserveerd (niet gebruiken) |
| \$v0 – \$v1 | 2 – 3 | Functie return waarde |
| \$a0 – \$a3 | 4 – 7 | Functie argumenten |
| \$t0 – \$t9 | 8 – 15, 24 – 25 | Temporary registers |
| \$s0 – \$s7 | 16 – 23 | Saved registers |
| \$k0 – \$k1 | 26 – 27 | Kernel Registers |
| \$gp | 28 | Pointer naar global data |
| \$sp | 29 | Stack pointer |
| \$fp | 30 | Frame pointer |
| \$ra | 31 | Return adres |
| \$f0 – \$f31 | | Floating point registers |

Labels

- Groeperen van instructies
- Vergelijkbaar met de naam van een functie (zie Python, C++)
- Maakt “jumps” en “branches” mogelijk

```
1 # add.asm: A program that computes the sum of 1 and 2
2 # leaving the result in register $t0.
3 # Registers used:
4 # t0 : used to hold the result
5 # t1 : used to hold the constant 1
6
7 main:                # start execution at main
8     li $t1, 1        # load value 1 into $t1
9     j add            # jump to label 'add'
10    addi $t0, $t1, 1  # $t0 = $t1 + 1
11 add:
12    addi $t0, $t1, 2  # $t0 = $t1 + 2
13 # end of add.asm
```

System calls

- Request aan besturingssysteem
- Functionaliteiten: input, output, memory, exit
- Hoe?
 1. Plaats code in register \$v0
 2. Roep syscall instructie op

System calls

| Naam | Code | Argumenten | Return register |
|--------------|------|--|-----------------|
| print_int | 1 | \$a0 | |
| print_float | 2 | \$f12 | |
| print_double | 3 | \$f12 | |
| print_string | 4 | \$a0 | |
| read_int | 5 | | \$v0 |
| read_float | 6 | | \$f0 |
| read_double | 7 | | \$f0 |
| read_string | 8 | \$a0 (geheugen adres) \$a1 (lengte) | |
| sbrk | 9 | \$a0 (lengte) | \$v0 adres |
| exit | 10 | | |

System calls

Voorbeeld

```
1 # add.asm: A program that computes the sum of 1 and 2
2 # Printing the result
3 # Registers used:
4 # t0 : used to hold the result
5 # t1 : used to hold the constant 1
6 main:
7     li    $t1, 1      # load 1 into $t1
8     addi $t0, $t1, 2 # $t0 = $t1 + 2
9
10    move $a0, $t0     # set result to $a0
11    li    $v0, 1      # load code for print_int
12    syscall
13 exit:
14    li    $v0, 10     # load code for exit
15    syscall
16 # end of add.asm
```

Geheugen

- Maakt het mogelijk om variabelen en data mee te geven in een script
- Data-gedeelte voorafgegaan door directive `.data`
- Voorbeelden:
 - `.ascii "abc": string`
 - `.asciiz "abc": string gevolgd door zero-byte (zie theorielessen)`
 - `.byte 5: 8-bit integer`
 - `.half -3: 16-bit integer`
 - `.word 3200: 32-bit integer`
 - `.space 20: lege ruimte, 20-bytes groot`
- Gebruik `lw` en `sw` instructies om data op te slaan en op te roepen
- Gebruik `la` om het adres te bekomen
- Instructie-gedeelte voorafgegaan door directive `.text`

Geheugen

Voorbeeld 1

```
1 # helloworld.asm: A "Hello World" program.
2 # Registers used:
3 # $v0 : syscall parameter and return value
4 # $a0 : syscall parameter: the string to print
5     .data
6     hello_msg: .asciiz "Hello World!\n"
7
8     .text
9     main:
10        la $a0, hello_msg    # load the addr of hello_msg in $a0
11        li $v0, 4            # load code for print_string
12        syscall
13     exit:
14        li $v0, 10          # load code for exit
15        syscall
```

Geheugen

Voorbeeld 2

```
1 # loadandstore.asm: Demonstrate load and store instructions
2 # by implementing c = a + b
3     .data
4 var_a: .word -5      # variable a
5 var_b: .word 8       # variable b
6 var_c: .word 0       # variable c
7
8     .text
9 main:
10     lw $t1, var_a    # load a in $t1
11     lw $t2, var_b    # load b in $t2
12     add $t0, $t1, $t2 # add a and b
13     sw $t0, var_c    # store sum into c
14
15
16 exit:
17     li $v0, 10       # load code for exit
18     syscall
```

Conditionele instructies

```
1 # conditional.asm
2 # c = max(a, b)
3     .data
4 var_a: .word 8      # variable a
5 var_b: .word 14     # variable b
6 var_c: .word 0      # variable c
7
8     .text
9 main:
10     lw $t1, var_a   # load a in $t1
11     lw $t2, var_b   # load b in $t2
12
13     #conditional: if a > b
14     bgt $t1, $t2, t1_greater # branch if $t1 > $t2
15     sw $t2, var_c   # store b into c
16     j endif         # jump to endif
17 t1_greater:
18     sw $t1, var_c   # store a into c
19 endif:
20     li $v0, 10      # load code for exit
21     syscall
```

Loops

```
1 # loop.asm
2 # c = a x b
3     .data
4 var_a:  .word 8      # variable a
5 var_b:  .word 5      # variable b
6 var_c:  .word 0      # variable c
7
8     .text
9 main:
10     lw $t1, var_a    # load a in $t1
11     lw $t2, var_b    # load b in $t2
12
13     #loop: add a to result, do this b times
14     li $t0, 0        # loop register
15     li $t3, 0        # result register
16 loop:
17     bge $t0, $t2, endloop    # end loop if loop register >= b
18     add $t3, $t3, $t1        # add a to result
19     addi $t0, $t0, 1        # increase loop register
20     j loop                  # jump to loop
21 endloop:
22     sw $t3, var_c          # store result into c
```

MARS Simulator

The screenshot shows the MARS Simulator window titled "/home/kasper/Desktop/MARS/add.asm - MARS 4.5". The interface includes a menu bar (File, Edit, Run, Settings, Tools, Help), a toolbar with various icons, and a main workspace divided into three sections.

Code Editor: Contains the following assembly code for "add.asm":

```
1 # add.asm: A program that computes the sum of 1 and 2
2 # Printing the result
3 # Registers used:
4 # $t0 : used to hold the result
5 # $t1 : used to hold the constant 1
6 main:
7     li    $t1, 1      # load 1 into $t1
8     addi $t0, $t1, 2 # $t0 = $t1 + 2
9
10    move $a0, $t0 # set result to $a0
11    li    $v0, 1      # load code for print_int
12    syscall
13
14    exit:
15     li    $v0, 10 # load code for exit
16     syscall
17 # end of add.asm
```

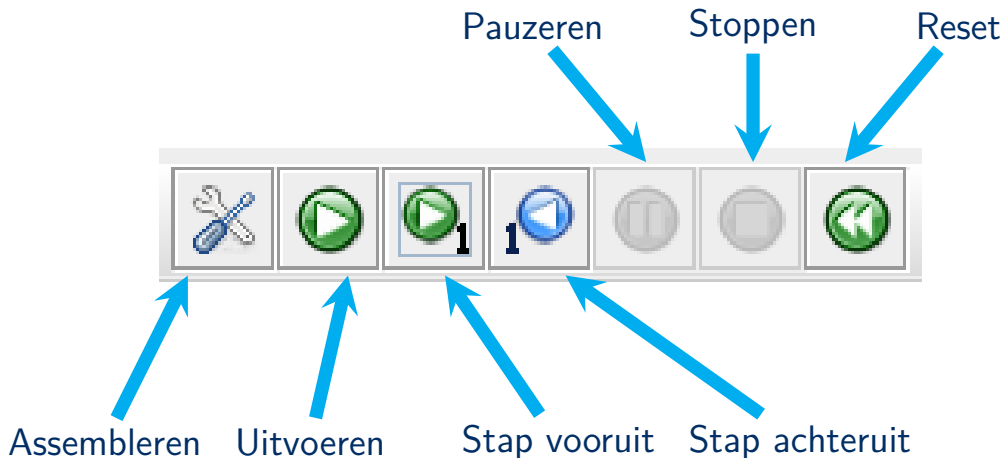
Registers Panel: A table showing the state of registers. The columns are "Name", "Number", and "Value".

| Name | Number | Value |
|--------|--------|------------|
| \$zero | 0 | 0x00000000 |
| \$at | 1 | 0x00000000 |
| \$v0 | 2 | 0x00000000 |
| \$v1 | 3 | 0x00000000 |
| \$a0 | 4 | 0x00000000 |
| \$a1 | 5 | 0x00000000 |
| \$a2 | 6 | 0x00000000 |
| \$a3 | 7 | 0x00000000 |
| \$t0 | 8 | 0x00000000 |
| \$t1 | 9 | 0x00000000 |
| \$t2 | 10 | 0x00000000 |
| \$t3 | 11 | 0x00000000 |
| \$t4 | 12 | 0x00000000 |
| \$t5 | 13 | 0x00000000 |
| \$t6 | 14 | 0x00000000 |
| \$t7 | 15 | 0x00000000 |
| \$s0 | 16 | 0x00000000 |
| \$s1 | 17 | 0x00000000 |
| \$s2 | 18 | 0x00000000 |
| \$s3 | 19 | 0x00000000 |
| \$s4 | 20 | 0x00000000 |
| \$s5 | 21 | 0x00000000 |
| \$s6 | 22 | 0x00000000 |
| \$s7 | 23 | 0x00000000 |
| \$t8 | 24 | 0x00000000 |
| \$t9 | 25 | 0x00000000 |
| \$k0 | 26 | 0x00000000 |
| \$k1 | 27 | 0x00000000 |
| \$gp | 28 | 0x10008000 |
| \$sp | 29 | 0x7fffffc |
| \$fp | 30 | 0x00000000 |
| \$ra | 31 | 0x00000000 |
| pc | | 0x00400000 |
| hi | | 0x00000000 |
| lo | | 0x00000000 |

Mars Messages Panel: Currently empty, with a "Clear" button at the bottom.

MARS Simulator

Toolbar



MARS Simulator

The screenshot displays the MARS Simulator window titled "/home/kasper/Desktop/MARS/add.asm - MARS 4.5". The interface includes a menu bar (File, Edit, Run, Settings, Tools, Help), a toolbar with various icons, and three main panels:

- Text Segment:** A table showing assembly instructions with columns for Bkpt, Address, Code, Basic, and Source. The first instruction is highlighted in yellow.
- Data Segment:** A table with columns for Address and Value, showing memory locations and their contents.
- Registers:** A table listing registers (Name, Number, Value) for Coproc 1 and Coproc 0.

The **Mars Messages** panel at the bottom shows the following output:

```
Assemble: assembling /home/kasper/Desktop/MARS/add.asm
Assemble: operation completed successfully.
```

A **Clear** button is located below the messages.

MARS Simulator

The screenshot displays the MARS Simulator window titled "/home/kasper/Desktop/MARS/add.asm - MARS 4.5". The interface includes a menu bar (File, Edit, Run, Settings, Tools, Help), a toolbar, and several panels:

- Text Segment:** A table showing assembly instructions with columns for Bkpt, Address, Code, Basic, and Source.

| Bkpt | Address | Code | Basic | Source |
|------------|------------|----------------------|-------|------------------------|
| 0x00400000 | 0x24090001 | addiu \$9,\$0,0x0... | 7: | li \$t1, 1 ... |
| 0x00400004 | 0x21280002 | addi \$8,\$9,0x0... | 8: | addi \$t0, \$t1, 2 ... |
| 0x00400008 | 0x00082021 | addu \$4,\$0,\$8 | 10: | move \$a0, \$t0 #... |
| 0x0040000c | 0x24020001 | addiu \$2,\$0,0x0... | 11: | li \$v0, 1 ... |
| 0x00400010 | 0x0000000c | syscall | 12: | syscall |
| 0x00400014 | 0x2402000a | addiu \$2,\$0,0x0... | 14: | li \$v0, 10 #... |
| 0x00400018 | 0x0000000c | syscall | 15: | syscall |
- Data Segment:** A table showing memory addresses and their values, currently all 0x000...

| Address | Value (...) | Value (...) | Value (...) | Value (...) | Value (...) | Value (...) | Value (...) | Value (...) |
|----------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| 0x100... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... |
| 0x100... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... |
| 0x100... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... |
| 0x100... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... |
| 0x100... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... |
| 0x100... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... |
| 0x100... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... |
| 0x100... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... |
| 0x100... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... | 0x000... |
- Registers:** A table listing registers, their numbers, and their values.

| Name | Number | Value |
|--------|--------|------------|
| \$zero | 0 | 0x00000000 |
| \$at | 1 | 0x00000000 |
| \$v0 | 2 | 0x0000000a |
| \$v1 | 3 | 0x00000000 |
| \$a0 | 4 | 0x00000003 |
| \$a1 | 5 | 0x00000000 |
| \$a2 | 6 | 0x00000000 |
| \$a3 | 7 | 0x00000000 |
| \$t0 | 8 | 0x00000003 |
| \$t1 | 9 | 0x00000001 |
| \$t2 | 10 | 0x00000000 |
| \$t3 | 11 | 0x00000000 |
| \$t4 | 12 | 0x00000000 |
| \$t5 | 13 | 0x00000000 |
| \$t6 | 14 | 0x00000000 |
| \$t7 | 15 | 0x00000000 |
| \$s0 | 16 | 0x00000000 |
| \$s1 | 17 | 0x00000000 |
| \$s2 | 18 | 0x00000000 |
| \$s3 | 19 | 0x00000000 |
| \$s4 | 20 | 0x00000000 |
| \$s5 | 21 | 0x00000000 |
| \$s6 | 22 | 0x00000000 |
| \$s7 | 23 | 0x00000000 |
| \$t8 | 24 | 0x00000000 |
| \$t9 | 25 | 0x00000000 |
| \$k0 | 26 | 0x00000000 |
| \$k1 | 27 | 0x00000000 |
| \$gp | 28 | 0x10008000 |
| \$sp | 29 | 0x7fffffc |
| \$fp | 30 | 0x00000000 |
| \$ra | 31 | 0x00000000 |
| pc | | 0x0040001c |
| hi | | 0x00000000 |
| lo | | 0x00000000 |
- Mars Messages:** A text area showing the output: "-- program is finished running --".

MIPS Reference sheet

- Overzicht van alle instructies en betekenis
 - Arithmetic, Logic instructions
 - Branch, Jump instructions
 - Memory instructions
- Te vinden op de MSDL-website