MODUL 5 : PCSPIM DAN BAHASA ASSEMBLY MIPS (Bagian-1)

Tujuan KerjaLab:
1. Mengetahui konsep dasar MIPS dan hubungannya dengan bahasa Assembly
2. Mengetahui jenis-jenis instruksi pada MIPS serta sintaksnya
3. Mengetahui cara menggunakan tools PCSPIM dalam menuliskan kode-kode bahasa Assembly

MIPS DAN BAHASA ASSEMBLY

Berdasarkan perancangan perangkat instruksinya, ada 2 jenis arsitektur prosesor yang menonjol saat ini, yaitu arsitektur RISC *(Reduce Instruction Set Computer)* dan CISC *(Complex Instruction Set Computer)*. Prosesor RISC memiliki instruksi-instruksi yang lebih sederhana dan lebih cepat untuk dieksekusi dibandingkan prosesor CISC.

Prosesor RISC memiliki ciri-ciri khusus, yaitu:

- Prosesor RISC mengeksekusi instruksi pada setiap satu siklus detak (Robinson, 1987 : 144; Johnson, 1987 : 153)
- Instruksi pada prosesor RISC memiliki format tetap, sehingga rangkaian pengontrol instruksi menjadi lebih sederhana
- Instruksi yang berhubungan dengan memori hanya instruksi isi *(load)* dan instruksi simpan *(store)*, instruksi lain dilakukan dalam register internal prosesor
- Prosesor RISC memerlukan waktu kompilasi yang lebih lama daripada prosesor CISC

MIPS *(Microprocessor without Interlocked Pipeline Stages)* merupakan salah satu contoh prosesor yang dibangung dengan arsitektur RISC. Desain prosesor MIPS saat ini banyak digunakan pada beberapa *embedded system* (seperti the Series2 TiVo, Windows CE devices, Cisco routers, residential gateways, Foneras, Avaya) dan *video games console* (seperti Nintendo 64 and Sony PlayStation, PlayStation 2, PlayStation Portable)

Bahasa Assembly merupakan bahasa pemrograman tingkat rendah *(Low Level Programming Language)* yang kita gunakan untuk memberikan instruksi-instruksi kepada prosesor MIPS. Untuk mensimulasikan pemrograman pada MIPS dengan bahasa Assembly, kita dapat menggunakan beberapa tools, salah satunya ialah PCSpim. Dengan PCSpim, kita dapat meng-*compile*, menjalankan, dan melihat hasil dari kode-kode program kita.

Namun, karena Assembly adalah bahasa tingkat rendah yang instruksinya terkait erat dengan bahasa mesin, maka penggunaan *resource*-nya pun sangat terbatas. Tidak seperti halnya pada bahasa pemrograman tingkat tinggi, jumlah operan pada instruksi-instruksi di bahasa Assembly MIPS sangatlah terbatas sesuai jumlah register yang mana digunakan sebagai tempat penyimpanan data. MIPS memiliki register sebanyak 32 buah dengan nomor 0 sampai 31. Untuk memudahkan

pengaksesan 32 register tersebut, kita dapat menuliskannya dengan simbol $ *(dollar)* yang diikuti dengan 2 buah karakter.

Register Number	Alternative Name	Description
0	zero	Constant 0
1	$at	(assembler temporary) -Reserved by the assembler
2 - 3	$v0 - $v1	(values) - From expression evaluation and function results
4 - 7	$a0 - $a3	(arguments) - First four parameters for subroutine.
		Not preserved across procedure calls
8 - 15	$t0 - $t7	(temporaries) - Caller saved if needed. Subroutines can use without saving.
		Not preserved across procedure calls
16 - 23	$s0 - $s7	(saved values) - Caller saved.
		A subroutine using one of these must save original and restore it before exiting.
		Preserved across procedure calls
24 - 25	$t8 - $t9	(temporaries) - Caller saved if needed. Subroutines can use without saving.
		These are in addition to $t0 - $t7 above.
		Not preserved across procedure calls.
26 - 27	$k0 - $k1	Reserved for use by the interrupt / trap handler
28	$gp	(global pointer)
		Points to the middle of the 64K block of memory in the static data segment.
29	$sp	(stack pointer)
		Points to last location on the stack.
30	$s8 - $fp	(saved value / frame pointer)
		Preserved across procedure calls
31	$ra	(return address)
		Used as the link register for jump and link instructions

INSTRUKSI PADA MIPS

Secara umum, instruksi pada MIPS dibagi menjadi 4 tipe, yaitu instruksi tipe Arithmetic Operation, Logical Operation, Data Transfer, dan Control, yang akan dijelaskan sebagai berikut.

- Instruksi tipe Arithmetic Operation

  Instruksi tipe ini adalah instruksi untuk operasi-operasi aritmatika, seperti penjumlahan, pengurangan, pembagian, perkalian, dan variasinya.

Instruksi	Operasi	Sintaks dan Contoh	Deskripsi
Add	$d = $s + $t	add $d, $s, $t	Menjumlahkan isi dari dua buah register dan menyimpan hasilnya ke register lain.
Add Immediate	$t = $s + imm	addi $t, $s, imm	Menjumlahkan isi sebuah register dengan sebuah *signed number* [imm] dan menyimpan hasilnya ke register lain
Add Imm. Unsigned	$t = $s + imm	addiu $t, $s, imm	Menjumlahkan isi sebuah register dengan sebuah *unsigned number* [imm] dan menyimpan hasilnya ke register lain
Substract	$d = $s - $t	sub $d, $s, $t	Mengurangkan isi dari dua buah register dan menyimpan hasilnya ke register lain.
Substract Unsigned	$d = $s - $t	subu $d, $s, $t	Mengurangkan isi dari dua buah register (tanpa memperhatikan tanda) dan menyimpan hasilnya ke register lain.
Divide	$LO = $s / $t; $HI = $s % $t	div $s, $t	Pembagian 2 buah register, menyimpan hasil bulatnya di $LO dan sisanya di $HI
Divide Unsigned	$LO = $s / $t; $HI = $s % $t	divu $s, $t	Pembagian 2 buah register, menyimpan hasil bulatnya di $LO dan sisanya di $HI
Move from High	$d = $HI	mfhi $d	Mengisi sebuah register dengan bilangan sisa pembagian atau operasi modulo ($HI)
Move from Low	$d = $LO	mflo $d	Mengisi sebuah register dengan bilangan bulat hasil operasi div ($LO)
Move	$s = $t	move $s, $t	Meng-*copy* isi sebuah register ke register lain
Multiply	$LO = $s * $t	mult $s, $t	Mengalikan 2 buah register dan menyimpan hasilnya ke $LO
Multiply Unsigned	$LO = $s * $t	multu $s, $t	Mengalikan 2 buah register dan menyimpan hasilnya ke $LO

❖ Instruksi tipe Logical Operation

Instruksi tipe ini meliputi operasi-operasi Boolean.

Instruksi	Operasi	Sintaks dan Contoh	Deskripsi
And	$1 = $2 & $3	and $1,$2,$3	Operasi AND dari 2 buah register
or	$1 = $2 | $3	or $1,$2,$3	Operasi OR dari 2 buah register
xor	$1 = $2 XOR $3	xor $1,$2,$3	Operasi XOR dari 2 buah register
nor	$1 = ~($2 | $3)	nor $1,$2,$3	Operasi NOR dari 2 buah register
and immediate	$1 = $2 & 10	andi $1,$2,10	Operasi AND antara sebuah register dengan angka

or immediate	$1 = $2 | 10	ori $1,$2,10	Operasi OR antara sebuah register dengan angka
xor immediate	$1 = ~$2 &~10	xori $1, $2,10	Operasi XOR antara sebuah register dengan angka st
shift left logical	$1 = $2 << 10	sll $1,$2,10	Geser ke kiri sebanyak nilai konstan yang diberikan
shift right logical	$1 = $2 >> 10	srl $1,$2,10	Geser ke kanan sebanyak nilai konstan yang diberikan
shift right arithmetic	$1 = $2 >> 10	sra $1,$2,10	Geser ke kanan sebanyak nilai konstan yang diberikan (*sign extended*)
shift left logical	$1 = $2 << $3	sllv $1,$2,$3	Geser ke kiri sebanyak nilai pada register
shift right logical	$1 = $2 >> $3	srlv $1,$2,$3	Geser ke kanan sebanyak nilai pada register
shift right arithmetic	$1 = $2 >> $3	srav $1,$2,$3	Geser ke kanan sebanyak nilai pada register (*sign extended*)

❖ Instruksi tipe Data Transfer

Instruksi tipe ini merupakan instruksi yang melibatkan pengambilan atau penyimpanan dari atau ke memori register.

Instruksi	Operasi	Sintaks dan Contoh	Deskripsi
Store Byte	MEM[$s + offset] = (0xff & $t)	sb $t, offset($s)	Least significant byte dari $t disimpan ke alamat yang ditentukan (offset($s))
Store Word	MEM[$s + offset] = $t	sw $t, offset($s)	Isi dari $t disimpan ke alamat yang ditentukan
Load Immediate	$s = imm	li $s, imm	Mengisi register $s dengan sebuah signed number [imm]
Load Byte	$t = MEM[$s + offset]	lb $t, offset($s)	Me-load sebuah byte ke sebuah register dari alamat yang ditentukan (offset($s))
Load Upper Immediate	$t = (imm << 16)	lui $t, imm	Sebuah angka [imm] digeser sebanyak 16 bit ke kiri dan disimpan ke dalam register
Load Word	$t = MEM[$s + offset]	lw $t, offset($s)	Me-load suatu nilai ke sebuah register dari alamat yang ditentukan (offset($s))

❖ Instruksi tipe Control

Instruksi	Operasi	Sintaks dan Contoh	Deskripsi
BEQ	if $s = $t; advance_pc (offset << 2))	beq $s, $t, offset	Jika $s dan $t sama, maka menuju ke alamat yang dituju

BNE	if $s != $t; advance_pc (offset << 2))	bne $s, $t, offset	Jika $s dan $t tidak sama, maka menuju ke alamat yang dituju
BGEZ	if $s >= 0; advance_pc (offset << 2))	bgez $s, offset	Menuju ke alamat yang dituju jika $s lebih besar atau sama dengan dari 0
BGEZAL	if $s >= 0; $31 = PC + 8 (or nPC + 4); advance_pc (offset << 2));	bgezal $s, offset	Menuju ke alamat yang dituju jika $s lebih besar atau sama dengan dari 0 dan menyimpan alamat tersebut ke $31
BGTZ	if $s > 0; advance_pc (offset << 2))	bgtz $s, offset	Menuju ke alamat yang dituju jika $s lebih besar dari 0
BLEZ	if $s <= 0; advance_pc (offset << 2))	blez $s, offset	Menuju ke alamat yang dituju jika $s lebih kecil atau sama dengan dari 0
BLTZ	if $s < 0; advance_pc (offset << 2))	bltz $s, offset	Menuju ke alamat yang dituju jika $s lebih kecil dari 0
BLTZAL	if $s < 0; $31 = PC + 8 (or nPC + 4); advance_pc (offset << 2));	bltzal $s, offset	Menuju ke alamat yang dituju jika $s lebih kecil atau sama dengan dari 0 dan menyimpan alamat tersebut ke $31
J	PC = nPC	j target	Melompat ke alamat target
JAL	$31 = PC + 8 (or nPC + 4)	jal target	Melompat ke alamat target dan menyimpan alamat tersebut ke $31
JR	PC = nPC; nPC = $s	jr $s	Melompat ke alamat yang merupakan isi dari register $s
SLT	if $s < $t; $d = 1; else $d = 0;	slt $d, $s, $t	Jika $s kurang dari $t, $d diset menjadi 1, dan 0 jika selainnya
SLTI	if $s < imm; $t = 1; else $t = 0;	slti $t, $s, imm	Jika $s kurang dari [imm], $t diset menjadi 1, dan 0 jika selainnya
SLTIU	if $s < imm; $t = 1; else $t = 0;	sltiu $t, $s, imm	Jika $s kurang dari unsigned [imm], $t diset menjadi 1, dan 0 jika selainnya
SLTU	if $s < $t; $d = 1; else $d = 0;	sltu $d, $s, $t	Jika $s kurang dari $t, $d diset menjadi 1, dan 0 jika selainnya

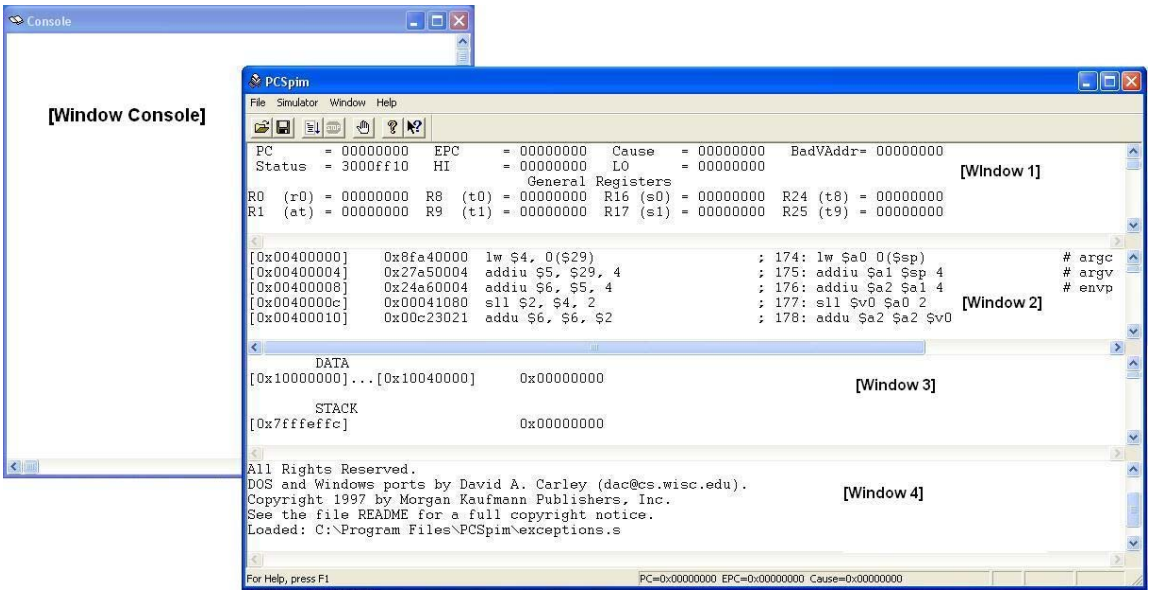
- System Call

System call merupakan sebuah interface yang menjembatani antara program dengan sistem operasi. System call dapat ditulis dalam bahasa Assembly atau bahasa tingkat tinggi yang dapat mengendalikan mesin. Berikut ini adalah system call yang terdapat pada MIPS.

Layanan	Nilai $v0	Argumen	Hasil
Print_int	1	$a0 = integer yang akan dicetak	
Print_float	2	$f12 = float yang akan dicetak	
Print_double	3	$f12 = double yang akan dicetak	
Print_string	4	$a0 = alamat string dalam memori	
Read_int	5		Integer dalam $v0
Read_float	6		Float dalam $v0
Read_double	7		Double dalam $v0
Read_string	8	$a0 = alamat memori buffer masukan string $a1 = panjang buffer string (n)	
Sbrk	9	$a0 = jumlah	Alamat dalam $v0
Exit	10		

PERKENALAN TOOLS PCSPIM

Setelah kita berhasil melakukan instalasi program PCSPIM di komputer kita, lalu kita jalankan program tersebut, maka akan muncul 2 window, yaitu window berlabel PCSPIM dan window Console, seperti yang tampak pada gambar berikut ini.



Keterangan:

Window PCSpim dibagi menjadi 4 window.

Window 1: window yang menampilkan register-register beserta *value* atau isinya

Window 2: window yang menampilkan *text segment* dan langkah-langkah atau tahapan eksekusi

Window 3: window yang menampilkan *data segment*

Window 4: window yang menampilkan *message* berkaitan dengan program yang sedang dieksekusi

Toolbar:

-  : Jalankan program yang sedang dibuka.
-  : Menghentikan program yang sedang dijalankan.
-  : Melakukan setting untuk menentukan letak Breakpoint.

Menubar Simulator:

- Clear Registers : Set nilai dari semua register ke *zero* (0x00000000).
- Reinitialize : Clears Registers dan memori, kemudian *restart* PCSpim termasuk file yang telah dibuka.
- Reload : Reinitializes simulator, kemudian *reload* yang sebelumnya dibuka.
- Go : Jalankan program yang sedang dibuka.
- Break/Continue : Jika program sedang jalan, *pause* eksekusi. Jika sedang di-*pause*, lanjutkan eksekusi.
- Single Step : Eksekusi satu baris instruksi.
- Multiple Step... : Eksekusi beberapa baris instruksi, jumlahnya ditentukan terlebih dahulu.
- Breakpoints... : Melakukan setting untuk menentukan letak Breakpoint.
- Set Value... : Melakukan setting *value* register yang ditunjuk oleh alamat tertentu.
- Display symbol table : Menampilkan symbol-simbol simulator ke window message.
- Settings... : Menampilkan kotak dialog Settings.

Window Console: untuk menampilkan hasil dari program kita.

STRUKTUR UMUM UNTUK MIPS ASSEMBLY LANGUAGE

```
# tanda pagar ini untuk menunjukkan komentar (tidak akan dieksekusi)

.data           # untuk mengawali pendeklarasian variabel (optional)

.text           # untuk mengawali instruksi-instruksi atau isi program (compulsory)

.globl main     # untuk menyatakan bahwa label main adalah label global yang akan dieksekusi
                  pertama kali dan yang menjadi acuan untuk program lainnya (compulsory)
```

Langkah-langkah pembuatan program

1. Kode-kode program dengan bahasa Assembly kita tuliskan di sebuah text editor, misalnya Notepad.
2. Simpan hasilnya dengan ekstensi .asm atau .s.
3. Kita buka kode program tadi dari PCSpim dengan File → Open.
4. Compile dan jalankan program dengan Simulator → Go atau tombol F5 atau tekan  pada toolbar.
5. Kita dapat melakukan tracing program dengan F10 (Single Step) atau F11 (Multiple Step).
6. Hasil dari program yang kita buat akan menulis output dan membaca inputan melalui window Console.

Melakukan debugging

- Kita dapat menentukan letak breakpoint dengan cara Simulator → Breakpoints. Masukkan *Address* dari instruksi yang kita inginkan menjadi tempat berhenti, lalu klik Add.
- Jika kita ingin mengeksekusi satu demi satu instruksi, kita dapat melakukannya dengan menekan tombol F10 (Simulator → Single Step)
- Jika kita ingin mengeksekusi beberapa instruksi secara bertahap, tekan F11 (Simulator → Multiple Step), lalu tentukan berapa banyak instruksi yang kita inginkan dalam sekali tahap.

IMPLEMENTASI KASUS-KASUS KECIL

- Deklarasi Data

  Format:
  name: storage_type value(s)

Catatan:

value(s) untuk memberikan nilai awal *(initial value)*

untuk tipe .space (string), diberikan ukuran untuk alokasi jumlah elemen

Contoh:

```
var1:  .word 3      # var1 bertipe integer bernilai awal 3
array1: .byte 'a','b' # array1 memiliki 2 elemen bertipe karakter
                # dengan nilai inisial 'a' dan 'b'
array2: .space 40     # dapat berupa 40 elemen array karakter
                # atau dapat berupa 40 elemen array integer
```

❖ Hello Word

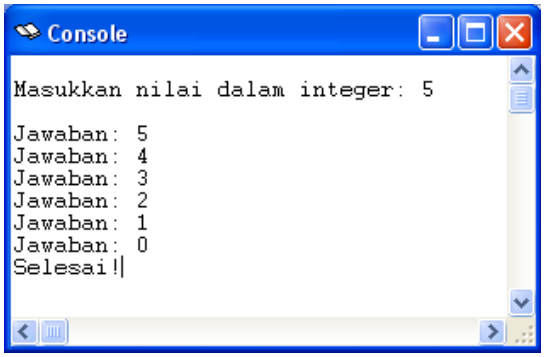
```
.data
 hello: .asciiz "Hello world!\n" # variabel string hello
.text
.globl main
main:
 la $a0, hello      # load address hello ke $a0
 li $v0, 4          # $v0, 4 -> perintah untuk print string
 syscall            # string hello di-print
exit:
 jr $ra             # mengakhiri program
```

❖ Input dan Output

```
.data
 teks1: .asciiz "\nJenengmu sopo? "      # '\n' untuk ganti baris
 teks2: .asciiz "\nSugeng rawuh, "
 jeneng: .space 100                      # .space untuk tipe array
.text
.globl main
main:
 la $a0, teks1
 li $v0, 4              # $v0, 4 -> perintah untuk print string
 syscall                # string teks1 di-print
 la $a0, jeneng
 li $v0, 8              # $v0, 8 -> perintah untuk membaca string
 syscall                # variabel jeneng diisi inputan nama
 la $a0, teks2
 li $v0, 4              # $v0, 4 -> perintah untuk print string
 syscall                # string teks2 di-print
 la $a0, jeneng
 li $v0, 4              # $v0, 4 -> perintah untuk print string
 syscall                # string jeneng di-print
exit:
 jr $ra                 # mengakhiri program
```

❖ Kondisional dan Perulangan

Tampilan yang diinginkan:




```
.data
  teks1: .asciiz "\nMasukkan nilai dalam integer: "
  teks2: .asciiz "\nJawaban: "
  teks3: .asciiz "\nSelesai!"
.text
.globl main
main:
  li $t1, 1                    # register $t1 diisi dengan nilai 1
  la $a0, teks1
  li $v0, 4                    # perintah untuk print string
  syscall                      # teks1 di-print
  li $v0, 5                    # perintah untuk baca integer
  syscall                      # sebuah bilangan integer dibaca
  move $t0, $v0                # copy isi register $v0 ke $t0
ulang:
  la $a0, teks2
  li $v0, 4                    # perintah untuk print string
  syscall                      # teks2 di-print
  move $a0, $t0                # copy isi register $t0 ke $a0
  li $v0, 1                    # perintah untuk print integer
  syscall                      # bilangan integer di $t0 di-print
  sub $t0, $t0, $t1            # $t0 = $t0 - 1
  beq $t0, $0, akhir           # jika $t0 = 0 maka lompat ke 'akhir'
  j ulang                      # balik ke 'ulang'
akhir
  la $a0, teks2
  li $v0, 4                    # perintah untuk print string
  syscall                      # teks2 di-print
  move $a0, $t0                # copy isi register $t0 ke $a0
  li $v0, 1                    # perintah untuk print integer
  syscall                      # bilangan integer di $t0 di-print
  la $a0, teks3
  li $v0, 4                    # perintah untuk print string
  syscall                      # teks3 di-print
exit:
  jr $ra                       # mengakhiri program
```