MISC CS TOPICS (IV)

Tony Wong

2016-01-30

Generations of Programming Language

	1st Generation Machine Code	2nd Generation Assembly	3rd Generation High-level	4th Generation Declarative
How to run	Can be directly run by CPU	Requires assembler	Requires compiler / interpreter	Requires interpreter
Portability	Depends on CPU / microarchitecture	Depends on architecture	High; Depends on compiler	High; Depends on interpreter
Languages			Pascal C/C++	SQL, Haskell

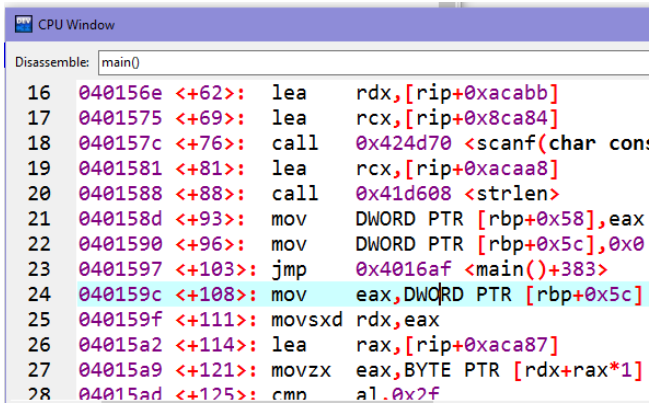
Example	0x24a60004
	0x0001d821
	0xac240204
		`lea eax,[rbp-0x60]`
		`sub eax,0x30`
		`jmp 0x4016ab`
			`int x = 0;`
			`while (n) {`
			`  x += n % 10;`
			`  n /= 10;`
			`}`
				`SELECT count(*) FROM`
				`` `users` WHERE `name` ``
				`LIKE 'CHAN%'`

Program and ISA

- A program is an ordered list of **instructions**
- Instructions are commands to the CPU
  - Perform calculations
  - Read data from / write data to memory
  - Control system components
- A **Instruction Set Architecture** defines
  - What instructions are available
  - What data types are available
  - How memory are addressed
  - I/O, Exception handling, Interrupt

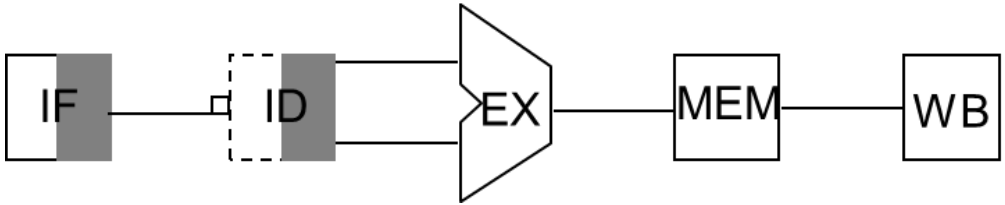
Assembly Language

- Low level language that is tightly coupled to a specific ISA
- Requires **assembler** to
  - Detect issues
  - Assign / convert addresses to symbols / labels
  - Convert pseudo-instructions into actual instructions
  - Convert instructions into binary
- Examples of ISAs
  - **x86**, **ARM**, AVR, PowerPC, 8051, MIPS
- A **disassembler** converts machine code back to assembly code



MIPS

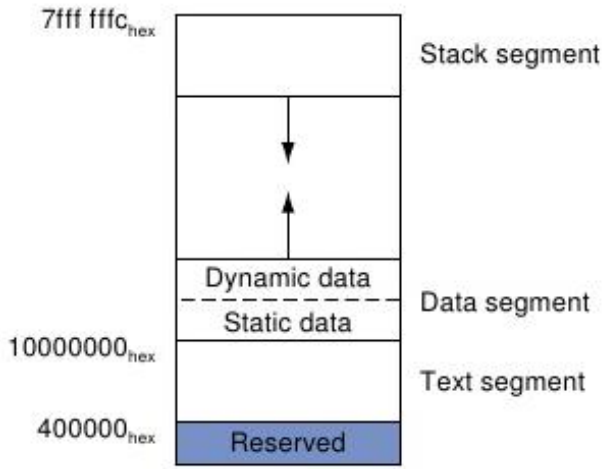
- Microprocessor without Interlocked Pipeline Stages
- MIPS was introduced in 1981
- We'll be using CPU "R2000" (introduced in 1985)
- 32 32-bit General Purpose **registers** (寄存器)
- Instructions are 32-bit too
  - In contrast, x86 instructions have variable length



helloworld.s

```
1          .data
2  msg:    .asciiz "Hello World\n"
3
4          .text
5  main:   li $v0, 4          # syscall 4 (print_str)
6          la $a0, msg        # argument: string
7          syscall            # print the string
8
9          jr $ra             # return
```

Assemblers, Linkers, and the SPIM Simulator
James R. Larus Microsoft Research

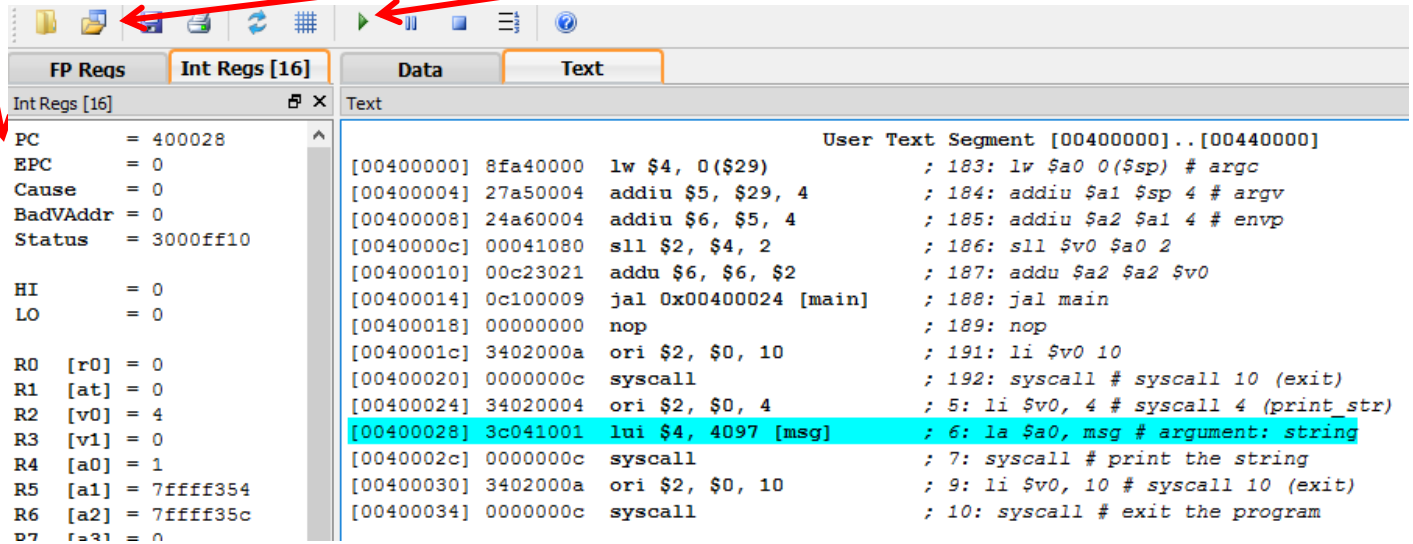


- Data segment: store constant data (e.g. strings) reserve space for variables / arrays
- Text segment: actual program
- The program will start from the **`main:`** label

SPIM (MIPS Simulator)

Program counter
(which instruction runs next)

EVERY TIME You want to run a program, press Reinitialize and Load File, then Run/Continue



Registers
("variables" visible by CPU)

Memory Address

Machine Code

Processed Code

Your Assembly Code

Data segment

- MIPS Assembler provides various directives to help you describe data easily

```
str1: .asciiz "abcdef\n"
```

  - Creates a null-terminated string with data “abcdef\n”
  - The memory location of the string is labelled with str1

```
char str1[] = "abcdef\n";
```

```
str2: .space 30
```

```
char str2[30];
```

```
primes: .word 2, 3, 5, 7, 11, 13
```

  - Creates an integer array of length 6, initialized with 2, 3, 5, 7, 11, 13
  - The memory location of the array is labelled with primes

```
array1: .word 0:10
```

```
int primes[] = {2, 3, 5, 7, 11, 13};
```

  - Creates an array of length 10
  - Each element is 4 bytes and is initialized to 0
  - The memory location of the array is labelled with array1

```
int array1[10] = {0};
```

Registers

- Registers are like “variables”
- Registers in MIPS are like 32-bit integers
- Some of them have special meaning
  - Arguments to function calls
  - Return value of function
  - Stack counter
- You can use $t0 - $t9, $s0 - $s7 freely
- $zero is a special register that is always 0
- Excess data has to be stored in memory

Assignment and addition

`li $rd, imm` (pseudo-instruction)

- Load Immediate
- Example: `li $t0, 5`

	Pascal	C / C++
	`$t0 := 5`	`$t0 = 5`

`move $rd, $rs`

- Example: `move $t0, $t1`

	`$t0 := $t1`	`$t0 = $t1`

`add $rd, $rs, $rt`

- Adding $rs and $rt and store result into $rd
- Example: `add $t0, $t1, $t2`

	`$t0 := $t1 + $t2`	`$t0 = $t1 + $t2`

`addi $rd, $rs, imm`

- Adding $rs and 16-bit signed imm and store result into $rd
- Example: `add $t0, $t1, 1`

	`$t0 := $t1 + 1`	`$t0 = $t1 + 1`

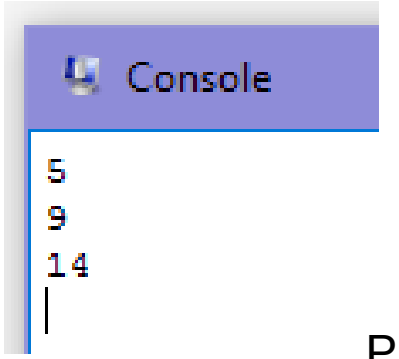
Syscalls

- Syscalls allows you to perform input and output
- Controlled by value stored in register $v0

Service	Code ($v0)	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8		$a0 = buffer, $a1 = length
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $a0)

aplusb.s

```
1         .data
2
3         .text
4   main: li $v0, 5       # syscall 5 (read_int)
5         syscall         # read integer
6         # at this point the input is stored in $v0
7         move $t0, $v0   # $t0 = $v0
8
9         li $v0, 5       # syscall 5 (read_int)
10        syscall         # read integer
11
12        add $a0, $t0, $v0  # $a0 = $t0 + $v0
13        li $v0, 1       # syscall 1 (print_int)
14        syscall         # $a0 is printed to screen
15
16        li $a0, 10      # ascii code of end line character
17        li $v0, 11      # syscall 11 (print_char)
18        syscall
19
20        jr $ra          # return
```



Pascal	C / C++
`readln($v0);`	`scanf("%d", $v0);`
`$t0 := $v0;`	`$t0 = $v0;`
`readln($v0);`	`scanf("%d", $v0);`
`$a0 := $t0 + $v0;`	`$a0 = $t0 + $v0;`
`write($a0);`	`printf("%d", $a0);`
`write(chr(10));`	`printf("%c", 10);`
	`return 0;`

Branch instructions

- Instructions normally execute from top to bottom
- To write “if” statements, we need branch instructions

`j label`
  - Unconditional jump
  - Example: `j loop`

`b?? $rs, label`
  - Conditional jump (if true)
  - Example: `beqz $t0, loop`

`b?? $rs, $rt, label` (pseudo-instruction)
  - Conditional jump (if true)
  - Example: `bgt $t0, $t1, loop`

Pascal	C / C++
`goto loop;`	`goto loop;`
`if ($t0 = 0) then goto loop;`	`if ($t0 == 0) goto loop;`
`if ($t0 > $t1) then goto loop;`	`if ($t0 > $t1) goto loop;`

phonenumber.s

```
1         .data
2 fixed:  .asciiz "Fixed\n"
3 mobile: .asciiz "Mobile\n"
4
5         .text
6 main:   li $v0, 5       # syscall 5 (read_int)
7         syscall         # read integer
8         # at this point the input is stored in $v0
9
10        la $a0, fixed         # assume its Fixed
11        li $t0, 40000000      # $t0 = 40000000
12        blt $v0, $t0, output # if ($v0 < $t0) goto output;
13        la $a0, mobile        # else: load mobile instead
14 output:
15        li $v0, 4       # syscall 4 (print_string)
16        syscall         # string pointed by $a0 is
17                        # printed to screen
18        jr $ra          # return
```

`la $rd, label`

- Load Address
- Example: **`la $t0, a`**
- Pascal: **`var $t0: Pointer;`**
  `$t0 := @a;`
- C/C++: **`void* $t0 = &a;`**

- Note that “false” part is above the “true” part

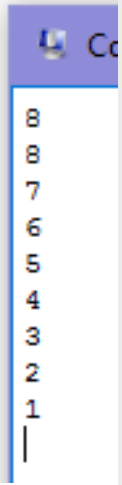
print_string syscall ($v0 = 4)

- C/C++: **`printf("%s", a0);`**

Loops

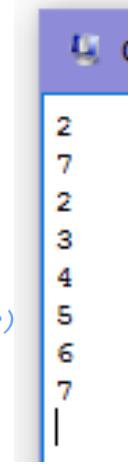
- do-while loop (dowhile.s)

```
1        .data
2
3        .text
4  main:
5        li $v0, 5
6        syscall
7        move $t0, $v0      # $t0 = $v0
8
9  loop:                    # do {
10       li $v0, 1          #   syscall 1 (print_int)
11       move $a0, $t0      #   $a0 = $t0
12       syscall
13
14       li $v0, 11         #   syscall 11 (print_char
15       li $a0, 10         #   end of line character
16       syscall
17
18       addi $t0, $t0, -1  #   $t0 = $t0 - 1
19       bgt $t0, $zero, loop # } while ($t0 > 0)
20
21       jr $ra             # return
```

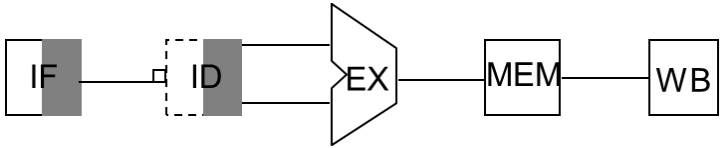


- for loop (forloop.s)

```
1        .data
2
3        .text
4  main:
5        li $v0, 5
6        syscall
7        move $t1, $v0      # $t1 is left
8        li $v0, 5
9        syscall
10       move $t2, $v0      # $t2 is right
11
12       move $t0, $t1      # int $t0 = $t1
13 loop:
14       bgt $t0, $t2, abc  # ; $t0 <= $t2;
15       li $v0, 1          #   syscall 1 (print_int)
16       move $a0, $t0      #   $a0 = $t0
17       syscall
18
19       li $v0, 11         #   syscall 11 (print_char)
20       li $a0, 10         #   end of line character
21       syscall
22
23       addi $t0, $t0, 1   #   $t0 = $t0 + 1
24       j loop             # unconditional jump
25
26 abc:   jr $ra             # return
```



Accessing Memory



- Unlike x86 and many other architectures MIPS is a load-store architecture
- Only the following instructions can access memory:
  - Load: `lw lh lb` Store: `sw sh sb`
  - `w h b` stands for 4 bytes, 2 bytes and 1 byte respectively
- Bare machine provides only one addressing mode: c(rx), which uses the sum of the immediate c and register rx as the address
- Addressing formats allowed by SPIM

Format	Address computation
(register)	contents of register
imm	immediate
imm (register)	immediate + contents of register
label	address of label
label ± imm	address of label + or – immediate
label ± imm (register)	address of label + or – (immediate + contents of register)

Strings

- syscall 8 (read_string) is similar to C function fgets($a0, $a1, stdin)
  - $a0 is the memory address of the buffer (where the string can be stored to)
  - $a1 is the length of the buffer

`lb $rd, offset($rs)`

- Load Byte (8-bit)
- Example: **`lb $t0, 5($t1)`**
- Pascal (t0 is char, t1 is ^char): **`$t0 := ($t1 + 5)^;`**
- C/C++ (t0 is char, t1 is char*): **`$t0 = $t1[5];`**

readstring.s

```
1         .data
2   str1: .space 30
3
4         .text
5   main:
6         li $v0, 8
7         la $a0, str1      # address of buffer
8         li $a1, 30        # size of buffer
9         syscall
10        # at this point the line is stored in address $a0
11
12        la $s0, str1
13
14        li $v0, 1         # syscall 1 (print_int)
15        lb $a0, 0($s0)    # get ascii code of $s0[0]
16        syscall
17
18        li $v0, 11        # syscall 11 (print_char)
19        li $a0, 10        # end of line character
20        syscall
21
22        li $v0, 1         # syscall 1 (print_int)
23        lb $a0, 2($s0)    # get ascii code of $s0[2]
24        syscall
25
26        li $v0, 11        # syscall 11 (print_char)
27        li $a0, 10        # end of line character
28        syscall
29
30        jr $ra
```

```
ABCD
65
67
```

Arrays

- Integers are 4-bytes wide
- To access an integer array, multiply index by 4 add then base address

$t2= 0x1000	0x1004	0x1008	0x100C	0x1010	0x1014
2	3	5	7	11	13

primes.s

```
1         .data
2  primes: .word 2, 3, 5, 7, 11, 13
3
4         .text
5  main:
6         li $t0, 0          # int i = 0;
7         li $t1, 6
8         la $t2, primes
9  loop:
10        beq $t0, $t1, abc # ; $t0 != $t1;
11
12        li $v0, 1          # syscall 1 (print_int)
13        mul $t3, $t0, 4    # $t3 = $t0 * 4 (number of bytes)
14        add $t4, $t2, $t3  # $t4 = $t2 + $t3 (memory address)
15        lw $a0, ($t4)      # get $t4[0]
16        syscall
17
18        li $v0, 11         # syscall 11 (print_char)
19        li $a0, 10         # end of line character
20        syscall
21
22        addi $t0, $t0, 1   # $t0 = $t0 + 1
23        j loop
24
25 abc:   jr $ra
```

```
2
3
5
7
11
13
```

`lw $rd, offset($rs)`

- Load Word (32-bit)
- Example: **`lw $t0, 8($t1)`**
- Pascal (t0 is longint, t1 is ^longint): **`$t0 := ($t1 + 2)^;`**
- C/C++ (t0 is int, t1 is int*): **`$t0 = $t1[2];`**

`sb $rd, offset($rs)` Store Byte **`sw $rd, offset($rs)`** Store Word

Pseudo instructions

- Instruction is only 32 bits. It is impossible load large values. Separate into two instructions (largenumber.s)

```
li $t0, 20000000
li $t1, 30000000

li $v0, 1
add $a0, $t0, $t1
syscall
```

```
[00400024] 3c010131  lui $1, 305              ; 5: li $t0, 20000000
[00400028] 34282d00  ori $8, $1, 11520
[0040002c] 3c0101c9  lui $1, 457              ; 6: li $t1, 30000000
[00400030] 3429c380  ori $9, $1, -15488
[00400034] 34020001  ori $2, $0, 1            ; 8: li $v0, 1
[00400038] 01092020  add $4, $8, $9           ; 9: add $a0, $t0, $t1
[0040003c] 0000000c  syscall                  ; 10: syscall
```

- Array access (primes2.s)

```
12          mul $t3, $t0, 4   # $t3 = $t0 + 4 (number of bytes)
13          lw $a0, primes + 0($t3) # get primes + $t3 bytes
```

```
[00400034] 34010004  ori $1, $0, 4            ; 12: mul $t3, $t0, 4 # $t3 = $t0 + 4 (number of bytes)
[00400038] 71015802  mul $11, $8, $1
[0040003c] 3c011001  lui $1, 4097             ; 13: lw $a0, primes + 0($t3) # get primes + $t3 bytes
[00400040] 002b0821  addu $1, $1, $11
[00400044] 8c240000  lw $4, 0($1)
```

Next Week 다음주

- Fast Forward 35 years
- Learn the most advanced technology today
- Write much faster programs
- Mini-comp again!