VE370 Mid Review

VE370 TA Group

Part 1: Context

- CPU Performance
- Assembly Basic
- Convention between Assembly and C
- Memory related operations
- Addressing
- Function Call Convention
- Linker

CPU Performance

- CPI, IC, Tc
- Pay attention to the unit

CPU Performance

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Clock Cycles per Instruction (CPI)}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

Classic CPU performance equation

$$\text{CPU Time} = \text{CPU Clock Cycles per program} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

Assembly Basic

- 4 principles:
  - Simplicity
  - Smaller
  - Fast common case
  - Compromise (similar formats)
- Basic Blocks
- How to load 32-bit Constants

Assembly Basic

- R-type, J-type, I-type
- Different positions of rs, rt, rd in MIPS vs. machine code

Assembly to C

- Array in C is like

	0	1	2	3
0	Array[0]			
4	Array[1]			
8	Array[2]			
C	Array[3]			

Assembly to C

- f = A[B[g]+1];

g: $s1

B: $s7

A: $s6

```
sll $t0, $s1, 2     # t0 = g*4
add $t0, $t0, $s7   # t0 += B
# t0 now is the address of B[g]

lw $t1, 0($t0)      # t1 = B[g]
addi $t1, $t1, 1    # B[g] + 1
# t1 now is B[g]+1

sll $t1, $t1, 2     # t1 * 4
add $t1, $t1, $s6   # t1 += A
# t1 now is the address of A[B[g]+1]

lw $s0, 0($t1)
```

Memory related

- lw, lh, lb: sign extended

Load Byte	lb	I	lw $s1,100($s2) $s1 = Mem[100+$s2]	R[rt]={24'b0, M[R[rs]+ZeroExtImm](7:0)}
Load Byte Unsigned	lbu	I		R[rt]={24'b0, M[R[rs]+SignExtImm](7:0)}
Load Halfword	lh	I	lui $s1,100 $s1 = 100*2^16	R[rt]={16'b0, M[R[rs]+ZeroExtImm](15:0)}
Load Halfword Unsigned	lhu	I		R[rt]={16'b0, M[R[rs]+SignExtImm](15:0)}
Load Upper Imm.	lui	I		R[rt]={imm,16'b0}
Load Word	lw	I		R[rt]=M[R[rs]+SignExtImm]
Load Immediate	li	P		R[rd]=immediate
Load Address	la	P		R[rd]=immediate

- E.g.

	0	1	2	3
0x10000000	15	47	89	33

lui $t0, 4096 # get the address 0x1000 0000

lb $s1, 2($t0)

Memory related

- lw, lh, lb: sign extended

Load Byte	lb	I	lw $s1,100($s2) $s1 = Mem[100+$s2]	R[rt]={24'b0, M[R[rs]+ZeroExtImm](7:0)}
Load Byte Unsigned	lbu	I		R[rt]={24'b0, M[R[rs]+SignExtImm](7:0)}
Load Halfword	lh	I	lui $s1,100 $s1 = 100*2^16	R[rt]={16'b0, M[R[rs]+ZeroExtImm](15:0)}
Load Halfword Unsigned	lhu	I		R[rt]={16'b0, M[R[rs]+SignExtImm](15:0)}
Load Upper Imm.	lui	I		R[rt]={imm,16'b0}
Load Word	lw	I		R[rt]=M[R[rs]+SignExtImm]
Load Immediate	li	P		R[rd]=immediate
Load Address	la	P		R[rd]=immediate

- E.g.

	0	1	2	3
0x10000000	15	47	89	33

lui $t0, 4096 # get the address 0x1000 0000

lb $s1, 2($t0) **-> 0xFFFF FF89**

Memory Related

- Big Endian: Most significant byte stored at least address of a word (MIPS is Big Endian)
- E.g. 0x12345678 in Big endian

0x10000000	0x10000001	0x10000002	0x10000003
12	34	56	78

- In Small endian

0x10000000	0x10000001	0x10000002	0x10000003
78	56	34	12

Addressing

- Immediate (e.g. addi)
- Register (e.g. add)
- Base (e.g. lw)
- PC-relative (Target address = new PC + offset * 4, new PC = PC +4)
- Pseudodirect (e.g. j and jal)
- (on L3 P19)

Addressing

```
Loop: slt $t2, $0, $t0
      bne $t2, $0, ELSE
      j Done
ELSE: addi $s2, $s2, 2
      addiu $t0, $t0, 1
      j Loop
Done:
```

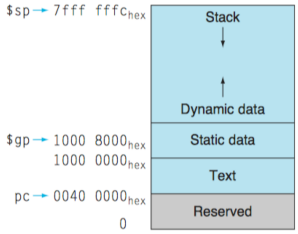
Address	Code
0x1000 0400	000000 00000 01000 01010 00000 101010 (R)
0x1000 0404	000101 01010 00000 0000000000000001 (I)
0x1000 0408	000010 00000000000000000100000110 (J)
0x1000 040c	001000 10010 1001000000000000010 (I)
0x1000 0410	001001 01000 0100000000000000001 (I)
0x1000 0414	000010 00000000000000000100000000 (J)

Function Call Convention

- Before Calling
  - change $sp to save $a0~$a3
  - Overwrite $a0~$a3
- During calling
  - Store $ra and any $s0~$s7 that will be used on stack
  - Execute and get $v0 $v1
  - Restore $s0~$s7
  - Restore(pop) frame
- After Calling
  - Restore $a0~$a3

Linker

- $gp initialized at center of static data
- Instructions stored in Text segment (from 0x0040 0000 towards up, won't exceed 0x1000 0000)
- Data stored in Static Data segment (from 0x1000 0000 towards up, won't exceed 0x1001 0000)



Linker

A

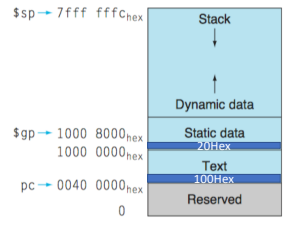
Object file header			
	Name	Procedure A	
	Text size	100_{hex}	
	Data size	20_{hex}	
Text segment	Address	Instruction	
	0	lw $a0, 0($gp)	
	4	jal 0	
	…	…	
Data segment	0	(X)	
	…	…	
Relocation information	Address	Instruction type	Dependency
	0	lw	X
	4	jal	B
Symbol table	Label	Address	
	X	–	
	B	–	

B

	Name	Procedure B	
	Text size	200_{hex}	
	Data size	30_{hex}	
	Address	Instruction	
	0	sw $a1, 0($gp)	
	4	jal 0	
	…	…	
	0	(Y)	
	…	…	
n	Address	Instruction type	Dependency
	0	sw	Y
	4	jal	A
	Label	Address	
	Y	–	
	A	–	

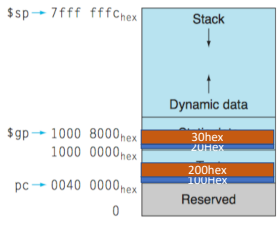
Linker, Cont'd

Object file header			
	Name	Procedure A	
	Text size	100_{hex}	
	Data size	20_{hex}	
Text segment	Address	Instruction	
	0	lw $a0, 0($gp)	
	4	jal 0	
	…	…	
Data segment	0	(X)	
	…	…	
Relocation information	Address	Instruction type	Dependency
	0	lw	X
	4	jal	B
Symbol table	Label	Address	
	X	–	
	B	–	



Linker, Cont'd

	Name	Procedure B	
	Text size	200_{hex}	
	Data size	30_{hex}	
	Address	Instruction	
	0	sw $a1, 0($gp)	
	4	jal 0	
	…	…	
	0	(Y)	
	…	…	
n	Address	Instruction type	Dependency
	0	sw	Y
	4	jal	A
	Label	Address	
	Y	–	
	A	–	



Linker, Cont'd A B

Object file header			
	Name	Procedure A	
	Text size	100_{hex}	
	Data size	20_{hex}	
Text segment	Address	Instruction	
	0	`lw $a0, 0($gp)`	
	4	`jal 0`	
	…	…	
Data segment	0	`(X)`	
	…	…	
Relocation information	Address	Instruction type	Dependency
	0	`lw`	X
	4	`jal`	B
Symbol table	Label	Address	
	X	–	
	B	–	

	Name	Procedure B	
	Text size	200_{hex}	
	Data size	30_{hex}	
	Address	Instruction	
	0	`sw $a1, 0($gp)`	
	4	`jal 0`	
	…	…	
	0	`(Y)`	
	…	…	
on	Address	Instruction type	Dependency
	0	`sw`	Y
	4	`jal`	A
	Label	Address	
	Y	–	
	A	–	

Linker, Cont'd A B

Object file header			
	Name	Procedure A	
	Text size	100_{hex}	
	Data size	20_{hex}	
Text segment	Address	Instruction	
	0	`lw $a0, 0($gp)`	
	4	`jal 0`	
	…	…	
Data segment	0	`(X)`	
	…	…	
Relocation information	Address	Instruction type	Dependency
	0	`lw`	X
	4	`jal`	B
Symbol table	Label	Address	
	X	–	
	B	–	

	Name	Procedure B	
	Text size	200_{hex}	
	Data size	30_{hex}	
	Address	Instruction	
	0	`sw $a1, 0($gp)`	
	4	`jal 0`	
	…	…	
	0	`(Y)`	
	…	…	
on	Address	Instruction type	Dependency
	0	`sw`	Y
	4	`jal`	A
	Label	Address	
	Y	–	
	A	–	

Linker, Cont'd A B

Object file header			
	Name	Procedure A	
	Text size	100_{hex}	
	Data size	20_{hex}	
Text segment	Address	Instruction	
	0	`lw $a0, 0($gp)`	
	4	`jal 0`	
	…	…	
Data segment	0	`(X)`	
	…	…	
Relocation information	Address	Instruction type	Dependency
	0	`lw`	X
	4	`jal`	B
Symbol table	Label	Address	
	X	–	
	B	–	

	Name	Procedure B	
	Text size	200_{hex}	
	Data size	30_{hex}	
	Address	Instruction	
	0	`sw $a1, 0($gp)`	
	4	`jal 0`	
	…	…	
	0	`(Y)`	
	…	…	
on	Address	Instruction type	Dependency
	0	`sw`	Y
	4	`jal`	A
	Label	Address	
	Y	–	
	A	–	

Linker, Cont'd A B

Object file header			
	Name	Procedure A	
	Text size	100_{hex}	
	Data size	20_{hex}	
Text segment	Address	Instruction	
	0	`lw $a0, 0($gp)`	
	4	`jal 0`	
	…	…	
Data segment	0	`(X)`	
	…	…	
Relocation information	Address	Instruction type	Dependency
	0	`lw`	X
	4	`jal`	B
Symbol table	Label	Address	
	X	–	
	B	–	

	Name	Procedure B	
	Text size	200_{hex}	
	Data size	30_{hex}	
	Address	Instruction	
	0	`sw $a1, 0($gp)`	
	4	`jal 0`	
	…	…	
	0	`(Y)`	
	…	…	
on	Address	Instruction type	Dependency
	0	`sw`	Y
	4	`jal`	A
	Label	Address	
	Y	–	
	A	–	

Linker Cont'd

Executable file header		
	Text size	300_{hex}
	Data size	50_{hex}
Text segment	Address	Instruction
	$0040\ 0000_{hex}$	`lw $a0, 8000hex($gp)`
	$0040\ 0004_{hex}$	`jal 40 0100hex`
	…	…
	$0040\ 0100_{hex}$	`sw $a1, 8020hex($gp)`
	$0040\ 0104_{hex}$	`jal 40 0000hex`
	…	…
Data segment	Address	
	$1000\ 0000_{hex}$	`(X)`
	…	…
	$1000\ 0020_{hex}$	`(Y)`
	…	…

2's complement

Absolute Address