


Administrative Matters

- **Quiz #1**
 - Wednesday, 10/9
 - Covers
 - Homework #1
 - Including today's lecture (10/8)
 - Textbook chapter 1 and 2
- **Homework #2**
 - Already posted, due Monday, 10/15
- **Quiz #2**
 - Wednesday 10/17
 - Cover Homework #2
 - All lecture till 10/15 (with emphasis on the untested part)
 - Textbook Chapter 1, 2, and 3 (with emphasis on 2 and 3)



COMPUTER SCIENCE & ENGINEERING

©2004 Morgan Kaufmann Publishers 1

Chapter 2

Instructions: Language of the Computer (continue)

COMPUTER SCIENCE & ENGINEERING

©2004 Morgan Kaufmann Publishers 2

MIPS Instruction Types

- simple instructions all 32 bits wide
- very structured, no unnecessary baggage
- only three instruction formats

| | | | | | | |
|---|----|----------------|----|----------------|-------|-------|
| R | op | rs | rt | rd | shamt | funct |
| I | op | rs | rt | 16 bit address | | |
| J | op | 26 bit address | | | | |

- rely on compiler to achieve performance
- In chapter 5, we will be implementing a MIPS processor

COMPUTER SCIENCE & ENGINEERING

©2004 Morgan Kaufmann Publishers 3

Addresses in Branches and Jumps

- **Instructions:**
 - bne \$t4,\$t5,Label Next instruction is at Label if \$t4!= \$t5
 - beq \$t4,\$t5,Label Next instruction is at Label if \$t4== \$t5
 - j Label Next instruction is at Label
- **Formats:**

| | | | | | | |
|---|----|----------------|----|----------------|--|--|
| I | op | rs | rt | 16 bit address | | |
| J | op | 26 bit address | | | | |
- **Addresses are 32 bits !**
 - How do we handle this with 16/26 bits?

COMPUTER SCIENCE & ENGINEERING

©2004 Morgan Kaufmann Publishers 4

Addresses in Branches

- **Instructions:**
 - bne \$t4,\$t5,Label Next instruction is at Label if \$t4!= \$t5
 - beq \$t4,\$t5,Label Next instruction is at Label if \$t4== \$t5
- **Formats:**

| | | | | | | |
|---|----|----|----|----------------|--|--|
| I | op | rs | rt | 16 bit address | | |
|---|----|----|----|----------------|--|--|
- use Instruction Address Register (PC = program counter)
 - Add it to the address
 - 16 bit signed integer (can branch back!!!)
 - Branch offset is always words (instruction are word aligned)
- most branches are local (principle of locality)
- **Spatial locality**
 - Object nearby will be accessed again
- **Temporal locality?**
 - Object most recently accessed will be accessed again
- Which one is this?

COMPUTER SCIENCE & ENGINEERING

©2004 Morgan Kaufmann Publishers 5

Addresses in Jump

- **Instructions:**
 - j Label Next instruction is at Label
- **Jump instructions just use high order bits of PC**
 - address boundaries of 256 M Byte (2²⁸ MByte)

| | | | | | |
|----|----------------|--|--|--|--|
| op | 26 bit address | | | | |
|----|----------------|--|--|--|--|
- **Question: Does PC has word address or byte address?**
- **Question: Is the 26 bit word offset or byte offset?**
- **Jump Register**
 - address entire 2³⁰ words (2³² bytes)
- **Question: Is this byte address, or word address?**
- **Question:**
 - 32bit can only address 4Gbyte, but I have a 160G disk, what gives?

COMPUTER SCIENCE & ENGINEERING

©2004 Morgan Kaufmann Publishers 6

Branch Offset Example

```

while (save[i]==k)
    i+=1;
    
```

```

Loop: sll $t1, $s3, 2    # Temp reg $t1 = 4 * i
    add $t1, $t1, $s6    # $t1 = address of save[i]
    lw $t0, 0($t1)      # Temp reg $t0 = save[i]
    bne $t0, $s5, Exit  # go to Exit if save[i]!=k
    addi $s3, $s3, 1    # i=i+1
    j Loop              # go to Loop
Exit:
    
```

- Loop at 80000
- bne use $2^4 + 80016 = 80024$
- j use absolute address
- $20000 \cdot 4 = 80000$

| | | | | | | | |
|-------|-----|----|----|-------|---|----|---|
| 80000 | 0 | 0 | 0 | 19 | 9 | 2 | 0 |
| 80004 | 0 | 9 | 22 | 9 | 0 | 32 | |
| 80008 | 35 | 9 | 8 | | | | 0 |
| 80012 | 5 | 8 | 21 | | | | 2 |
| 80016 | 8 | 19 | 19 | | | | 1 |
| 80020 | 2 | | | 20000 | | | |
| 80024 | ... | | | | | | |

©2004 Morgan Kaufmann Publishers 7

Branching Far Away

- beq \$s0, \$s1, L1
- If L1 turns out to be far away (than 16 bits allows)
- Use

```

    bne $s0, $s1, L2
    j L1
L2:
    
```

©2004 Morgan Kaufmann Publishers 8

MIPS address mode

Mistake in book

©2004 Morgan Kaufmann Publishers 9

MIPS summary

| MIPS operands | | |
|-----------------------|--|--|
| Name | Example | Comments |
| 32 registers | \$s0-\$s7, \$t0-\$t9, \$zero, \$a0-\$a3, \$v0-\$v1, \$gp, \$fp, \$sp, \$ra, \$at | Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register \$zero always equals 0. Register \$at is reserved for the assembler to handle large constants. |
| 2^{20} memory words | Memory[0], ..., Memory[4294967292] | Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls. |

This is for both code and data (store program concept)

©2004 Morgan Kaufmann Publishers 10

MIPS summary

| MIPS assembly language | | | | |
|-----------------------------|-------------------------|----------------------|---|-----------------------------------|
| Category | Instruction | Example | Meaning | Comments |
| Arithmetic | add | add \$s1, \$s2, \$s3 | $s1 = s2 + s3$ | Three operands; data in registers |
| | subtract | sub \$s1, \$s2, \$s3 | $s1 = s2 - s3$ | Three operands; data in registers |
| | add immediate | addi \$s1, \$s2, 100 | $s1 = s2 + 100$ | Used to add constants |
| Data transfer | load word | lw \$s1, 100(\$s2) | $s1 = \text{Memory}[s2 + 100]$ | Word from memory to register |
| | store word | sw \$s1, 100(\$s2) | $\text{Memory}[s2 + 100] = s1$ | Word from register to memory |
| | load byte | lb \$s1, 100(\$s2) | $s1 = \text{Memory}[s2 + 100]$ | Byte from memory to register |
| | store byte | sb \$s1, 100(\$s2) | $\text{Memory}[s2 + 100] = s1$ | Byte from register to memory |
| Conditional branch | branch on equal | beq \$s1, \$s2, 25 | $\{s1 == s2\} \rightarrow \text{go to PC} + 4 + 100$ | Equal test; PC-relative branch |
| | branch on not equal | bne \$s1, \$s2, 25 | $\{s1 \neq s2\} \rightarrow \text{go to PC} + 4 + 100$ | Not equal test; PC-relative |
| | set on less than | slt \$s1, \$s2, \$s3 | $\{s2 < s3\} \rightarrow s1 = 1; \text{ else } s1 = 0$ | Compare less than; for beq, bne |
| Unconditional jump | set less than immediate | slti \$s1, \$s2, 100 | $\{s2 < 100\} \rightarrow s1 = 1; \text{ else } s1 = 0$ | Compare less than constant |
| | jump | j 2500 | go to 10000 | Jump to target address |
| Unconditional jump and link | jump register | jr \$ra | go to \$ra | For switch, procedure return |
| | jump and link | jal 2500 | $ra = PC + 4; \text{ go to } 10000$ | For procedure call |

©2004 Morgan Kaufmann Publishers 11

Quick Questions

- What is the range of addresses for conditional branches in MIPS ($K=1024$)
2¹⁰
 - Address between 0 and 64K-1
 - Address between 0 and 256K-1
 - Addresses from about 32K before the branch to 32K after
 - Addresses from about 128K before the branch to 128K after
- What is the range of address from jump and jal in MIPS ($M=1024K$)
 - Address between 0 and 64M-1
 - Address between 0 and 256M-1
 - Addresses from about 32M before the branch to about 32M after
 - Addresses from about 128M before the branch to about 128M after
 - Anywhere within a block of 64M addresses where the pc supply the upper 6 bits
 - Anywhere within a block of 256M address where the PC supply the upper 4 bits

©2004 Morgan Kaufmann Publishers 12

Alternative Architectures

- MIPS instruction
 - Very fast cycle time
 - A few instruction type
 - A few kinds of instruction
 - Often lead to very high number of instructions executed
- Design alternative:
 - provide more powerful operations
 - goal is to reduce number of instructions executed
 - danger is a slower cycle time and/or a higher CPI
- Let's look (briefly) at IA-32

COMPUTER SCIENCE & ENGINEERING
©2004 Morgan Kaufmann Publishers 13

IA - 32

- 1978: The Intel 8086 is announced (16 bit architecture)
- 1980: The 8087 floating point coprocessor is added
- 1982: The 80286 increases address space to 24 bits, +instructions
- 1985: The 80386 extends to 32 bits, new addressing modes
- 1989-1995: The 80486, Pentium, Pentium Pro add a few instructions (mostly designed for higher performance)
- 1997: 57 new "MMX" instructions are added, Pentium II (SIMD)
- 1999: The Pentium III added another 70 instructions (SSE)
 - Streaming SIMD Extension
- 2001: Another 144 instructions (SSE2)
- 2003: AMD extends the architecture to increase address space to 64 bits, widens all registers to 64 bits (AMD64)
- 2004: Intel capitulates and embraces AMD64 (calls it EM64T) and adds more media extensions (SSE3)
 - Extended Memory 64 Technology

COMPUTER SCIENCE & ENGINEERING
©2004 Morgan Kaufmann Publishers 14

IA-32 Overview

- Complexity:
 - Instructions can vary in length
 - one operand must act as both a source and destination
 - one operand can come from memory
 - complex addressing modes
e.g., "base or scaled index with 8 or 32 bit displacement"
- What does MIPS do, and why?

COMPUTER SCIENCE & ENGINEERING
©2004 Morgan Kaufmann Publishers 15

IA-32 Overview

- Saving grace:
 - the most frequently used instructions are not too difficult to build
 - compilers avoid the portions of the architecture that are slow
 - can't argue with success
 - Pretty much all desktop machines are IA-32 based
 - (though there are much more embedded computers)
- Philosophical question:
 - Will IA-32/64 continue to dominate the landscape?
 - Why?

COMPUTER SCIENCE & ENGINEERING
©2004 Morgan Kaufmann Publishers 16

IA-32 Registers

- Registers in the 32-bit subset that originated with 80386

COMPUTER SCIENCE & ENGINEERING
©2004 Morgan Kaufmann Publishers 17

IA-32 Instruction Types

- Instruction Types for Arithmetic, Logic, and Data Transfer

| Source/Destination Operand type | Second Source Operand |
|---------------------------------|-----------------------|
| Register | Register |
| Register | Immediate |
| Register | Memory |
| Memory | Register |
| Memory | Immediate |

COMPUTER SCIENCE & ENGINEERING
©2004 Morgan Kaufmann Publishers 18

IA-32 Addressing Modes

| Mode | Description | Register restrictions | MIPS equivalent |
|---|--|----------------------------------|---|
| Register indirect | Address is in a register. | not ESP or EBP | lw \$s0, 0(\$s1) |
| Based mode with 8- or 32-bit displacement | Address is contents of base register plus displacement. | not ESP or EBP | lw \$s0, 100(\$s1) # $\le 16\text{-bit}$ # displacement |
| Base plus scaled index | The address is $\text{Base} + (2^{\text{Scale}} \times \text{Index})$ where Scale has the value 0, 1, 2, or 3. | Basic: any GPR Index: not ESP | rdi \$t0, \$s2, 4 add \$t0, \$t0, \$s1 lw \$s0, 0(\$t0) |
| Base plus scaled index with 8- or 32-bit displacement | The address is $\text{Base} + (2^{\text{Scale}} \times \text{Index}) + \text{displacement}$ where Scale has the value 0, 1, 2, or 3. | Basic: any GPR Index: not ESP | rdi \$t0, \$s2, 4 add \$t0, \$t0, \$s1 lw \$s0, 100(\$t0) # $\le 16\text{-bit}$ # displacement |

Addressing mode for an operand in IA32 = several MIPS instructions

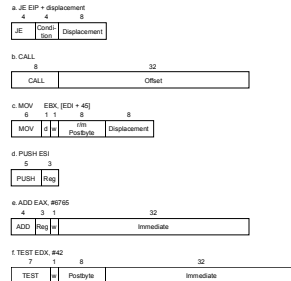
IA-32 Typical Instructions

- Four major types of integer instructions:
 - Data movement including move, push, pop
 - Arithmetic and logical (destination register or memory)
 - Control flow (use of condition codes / flags)
 - String instructions, including string move and string compare
- What does MIPS do? Why?

| Instruction | Function |
|--------------------|--|
| JE name | if (equal) (condition code) (EIP=name); EIP = 32-bit name \ll EIP+128 |
| JMP name | EIP=name |
| CALL name | SP=SP-4; M[SP]=EIP+5; EIP=name |
| MOVW EBX, [EDI+45] | EBX=M[EDI+45] |
| PUSH ESI | SP=SP-4; M[SP]=ESI |
| POP EDI | EDI=M[SP]; SP=SP+4 |
| ADD EAX, #6765 | EAX=EAX+6765 |
| TEST EDX, #42 | Set condition code (flags) with EDX and 42 |
| MOVSL | M[EDI]=M[ESI]; EDI=EDI+4; ESI=ESI+4 |

IA-32 instruction Formats

- Jump based on condition code
- Instruction encoding using postbyte



Summary

- Design Principles:
 - simplicity favors regularity
 - smaller is faster
 - make the common case fast
- Instruction set architecture
 - Arithmetic, logical, conditionals, procedures
 - Registers, memory, constants
 - Operands, addressing mode
 - Stacks
 - MIPS vs. IA-32