Instruction Set Architecture II

Instructor: Dmitri A. Gusev

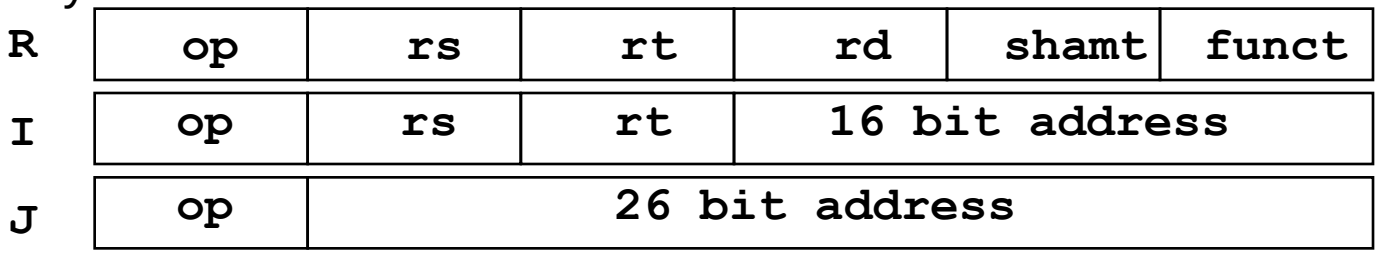
Fall 2007

CS 502: Computers and Communications

Lecture 3, September 12, 2007

Overview of MIPS

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



- rely on compiler to achieve performance
  — what are the compiler's goals?
- help compiler where we can

Constants

- Small constants are used quite frequently (50% of operands)

  e.g., A = A + 5;
  B = B + 1;
  C = C - 18;

- Solutions? Why not?
  - put 'typical constants' in memory and load them.
  - create hard-wired registers (like $zero) for constants like one.

- MIPS Instructions:

```
addi $29, $29, 4
slti $8, $18, 10
andi $29, $29, 6
ori $29, $29, 4
```

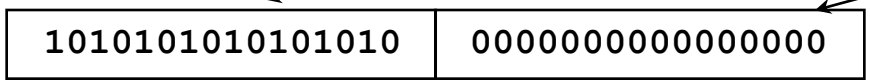
- Design Principle: Make the common case fast.

How about larger constants?

- We'd like to be able to load a 32 bit constant into a register
- Must use two instructions, new "load upper immediate" instruction

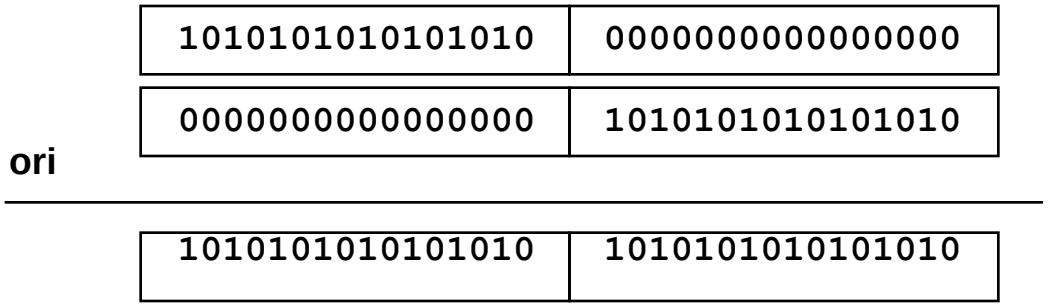
```
lui $t0, 1010101010101010
```

filled with zeros



- Then must get the lower order bits right, i.e.,

```
ori $t0, $t0, 1010101010101010
```



PC-Relative Addressing

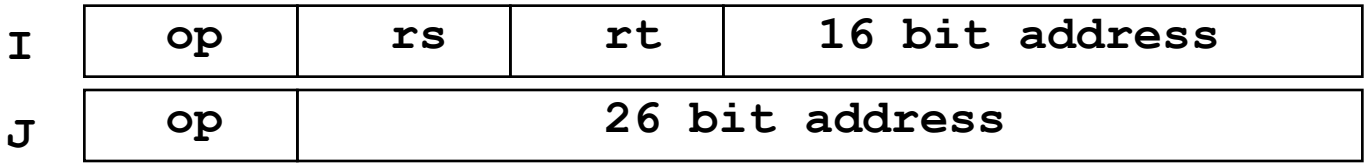
- PC-relative addressing is an addressing regime in which the address is the sum of the program counter (PC) and a constant in the instruction

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:



- Addresses are not 32 bits
  - — How do we handle this with load and store instructions?

Addresses in Branches

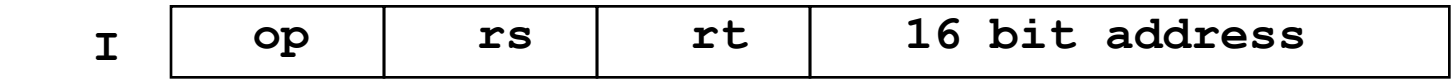
- Instructions:

```
bne $t4,$t5,Label
beq $t4,$t5,Label
```

Next instruction is at Label if $t4≠$t5

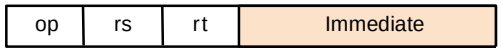
Next instruction is at Label if $t4=$t5

- Formats:

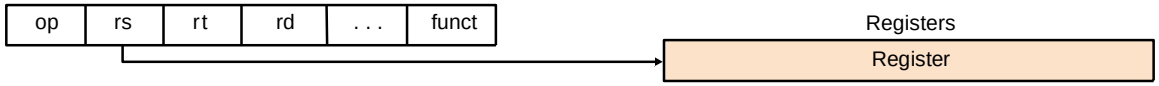


- Could specify a register (like lw and sw) and add it to address
  - use Instruction Address Register (PC = program counter)
  - most branches are local (principle of locality)
- Jump instructions just use high order bits of PC
  - address boundaries of 256 MB

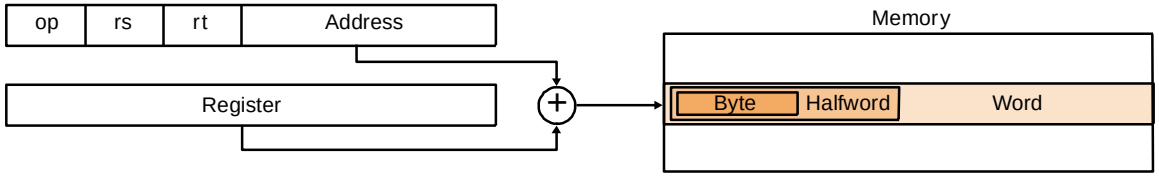
1. Immediate addressing



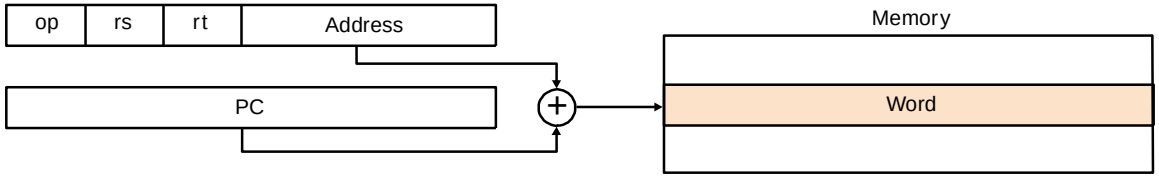
2. Register addressing



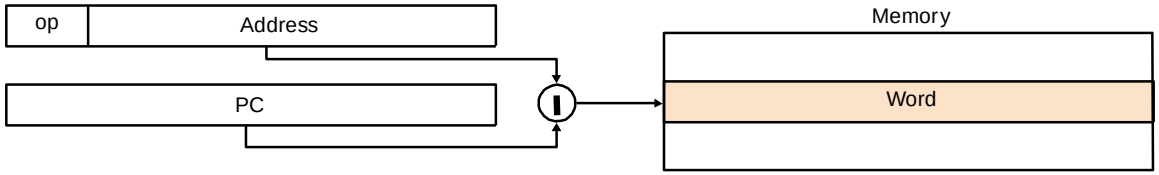
3. Base addressing



4. PC-relative addressing



5. Pseudodirect addressing



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
- 1999: The Pentium III added another 70 instructions (SSE)
- 2001: Another 144 instructions (SSE2)
- 2003: AMD extends the architecture to increase address space to 64 bits, widens all registers to 64 bits and other changes (AMD64)
- 2004: Intel capitulates and embraces AMD64 (calls it EM64T) and adds more media extensions

- “This history illustrates the impact of the “golden handcuffs” of compatibility

  “adding new features as someone might add clothing to a packed bag”

  “an architecture that is difficult to explain and impossible to love”

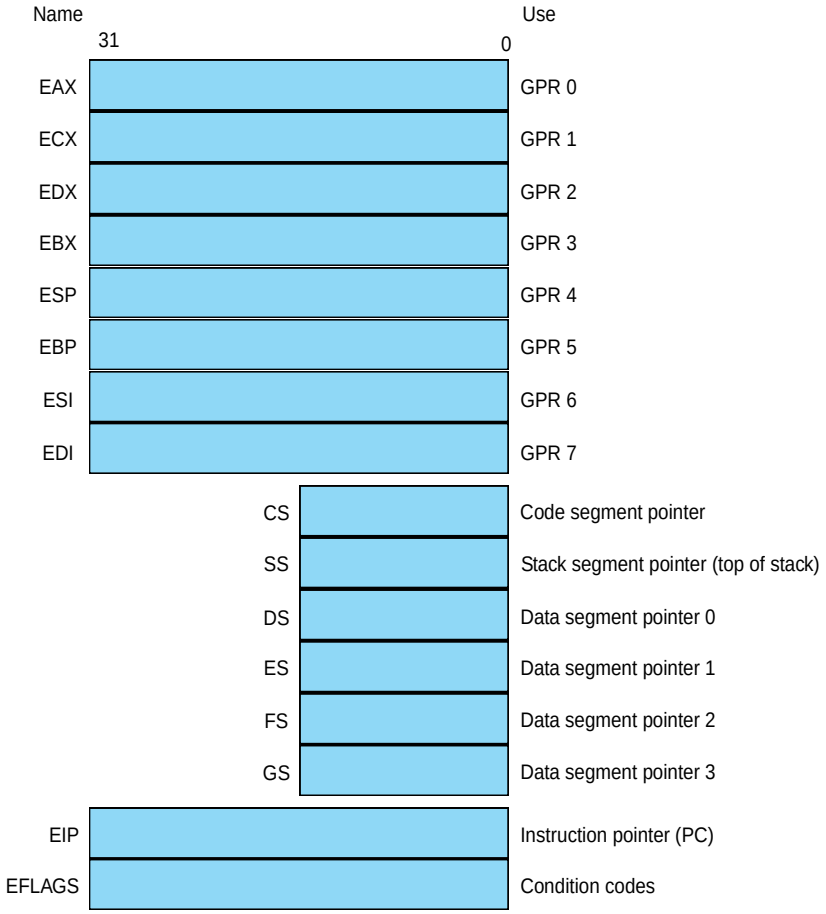
IA-32 Overview

- Complexity:
  - Instructions from 1 to 17 bytes long
  - 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”
- Saving grace:
  - the most frequently used instructions are not too difficult to build
  - compilers avoid the portions of the architecture that are slow

“what the 80x86 lacks in style is made up in quantity, making it beautiful from the right perspective”

IA-32 Registers and Data Addressing

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



IA-32 Register Restrictions

- Registers are not “general purpose” – note the restrictions below

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)#≤16-bit #displacement`

Base plus scaled index	The address is Base + (2^{Scale} x Index) where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4`
			`add $t0,$t0,$s1`
			`lw $s0,0($t0)`

Base plus scaled index with 8- or 32-bit displacement	The address is Base + (2^{Scale} x Index) + displacement where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4`
			`add $t0,$t0,$s1`
			`lw $s0,100($t0)#≤16-bit #displacement`

FIGURE 2.42 IA-32 32-bit addressing modes with register restrictions and the equivalent MIPS code. The Base plus Scaled Index addressing mode, not found in MIPS or the PowerPC, is included to avoid the multiplies by four (scale factor of 2) to turn an index in a register into a byte address (see Figures 2.34 and 2.36). A scale factor of 1 is used for 16-bit data, and a scale factor of 3 for 64-bit data. Scale factor of 0 means the address is not scaled. If the displacement is longer than 16 bits in the second or fourth modes, then the MIPS equivalent mode would need two more instructions: a `lui` to load the upper 16 bits of the displacement and an `add` to sum the upper address with the base register `$s1`. (Intel gives two different names to what is called Based addressing mode—Based and Indexed—but they are essentially identical and we combine them here.)

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

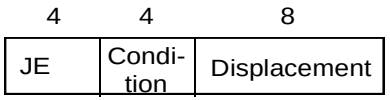
Instruction	Function
JE name	if equal(condition code) {EIP=name}; EIP-128 ≤ name < EIP+128
JMP name	EIP=name
CALL name	SP=SP-4; M[SP]=EIP+5; EIP=name;
MOVW EBX,[EDI+45]	EBX=M[EDI+45]
PUSH ESI	SP=SP-4; M[SP]=ESI
POP EDI	EDI=M[SP]; SP=SP+4
ADD EAX,#6765	EAX= EAX+6765
TEST EDX,#42	Set condition code (flags)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

FIGURE 2.43 Some typical IA-32 instructions and their functions. A list of frequent operations appears in Figure 2.44. The CALL saves the EIP of the next instruction on the stack. (EIP is the Intel PC.)

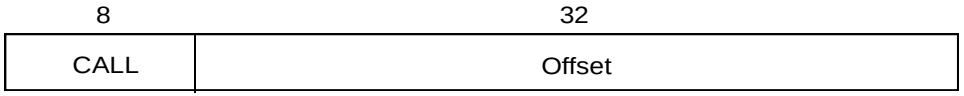
IA-32 instruction Formats

- Typical formats: (notice the different lengths)

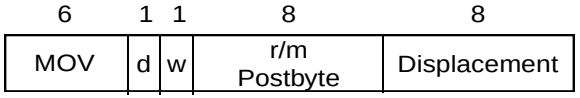
a. JE EIP + displacement



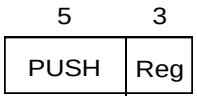
b. CALL



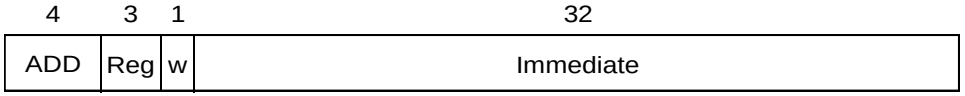
c. MOV EBX, [EDI + 45]



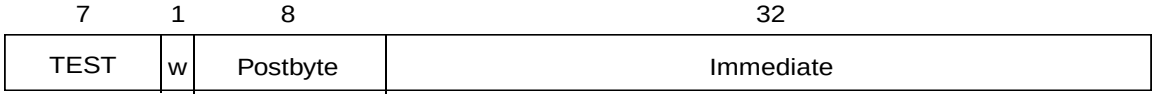
d. PUSH ESI



e. ADD EAX, #6765



f. TEST EDX, #42



Summary

- Instruction complexity is only one variable
  - lower instruction count vs. higher CPI / lower clock rate
- Design Principles:
  - simplicity favors regularity
  - smaller is faster
  - good design demands compromise
  - make the common case fast
- Instruction set architecture
  - a very important abstraction indeed!