Outline

- A look at other instruction sets
  - PowerPC
  - Intel 80x86

Power PC 1/3

- Another RISC example (made by IBM and Motorola, and used in Apple Macintosh)
- 32 integer registers, instructions 32 bits long
- Two more addressing modes

Indexed addressing

Allow 2 registers to be added together, similar to base address and offset (array index)

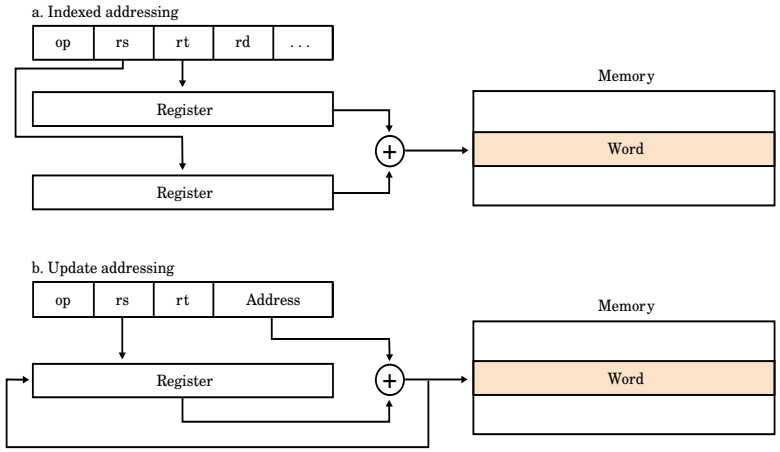
```
add $t0, $a0,  $s3        →    lw $t1,$a0+$s3
lw $t1, 0($t0)
```

Update addressing

Automatically increment the base register to point to next word each time data is transferred

```
lw $t0,4($s3)             →    lwu $t0,4($s3)
addi $s3,$s3,4
```


Power PC 2/3



Power PC 3/3

- Load multiple and store multiple

  Transfer up to 32 words of data in a single instruction

- Special counter register, separate from the other 32 registers, to try to improve performance of a *for* loop

In MIPS

Loop:

```
addi $t0,$t0,-1          #$t0 = $t0-1
bne $t0,$zero,Loop       # if $t0 != 0 go to Loop
```

In Power PC

```
bc Loop,ctr != 0         # $ctr = $ctr-1;
                         # if $ctr != 0 go to Loop
```

Intel 80x86 1/7

1978 Intel 8086 (assembly language extension to 8080 (8-bit)) 16 bit architecture, all internal registers 16 bits. Dedicated uses not considered general-purpose register architecture

1980 Intel 8087 (floating-point coprocessor) (relies on a stack and not registers)

1982 80286 (address space 24 bits)

1985 80386 (32-bits architecture, 32-bits registers, 32-bits address space)

1989-1995 80486, Pentium, and Pentium Pro

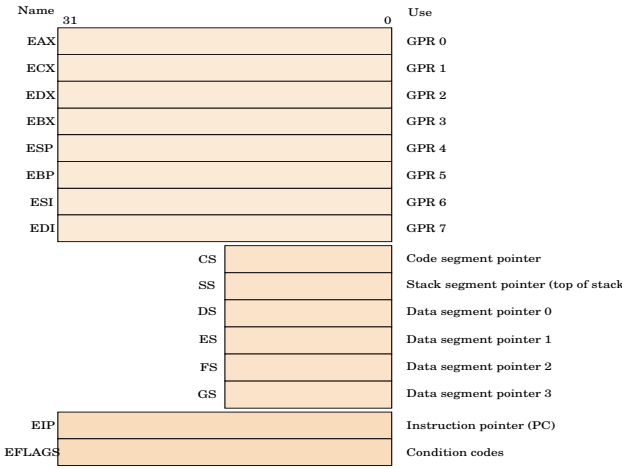
1997 MMX architecture, uses floating-point stack to accelerate multimedia and communication applications.

Pentium II

Pentium III

Pentium 4 (2.8 GHz Aug. 2002)

Intel 80x86 2/7



80386 register set

Intel 80x86 3/7

•Instruction types for arithmetic, logical, and data transfer instructions (two-operand instructions)

Source/Destination operand	Second source operand
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate

•Must have one operand that acts as both source/destination (MIPS allows separate registers for source and destination)

•One of the operands can be in memory but not both

Intel 80x86 4/7

•Addressing Modes

Mode	Description	MIPS equivalent
Register Indirect	Address is in a register	lw $s0,0($s1)
Based mode with 8 or 32-bit displacement	Address is contents of base register plus displacement	lw $s0,100($s1) (16-bit displacement)

Base plus scaled Index	Address is Base + (2^{scale} * index)
	Where scale can be 0, 1, 2, or 3
	scale = 0, address not scaled
	Scale =1, 16-bit data
		mul $t0,$s2,4
		add $t0,$t0,$s1
		lw $s0,0($t0)
		(scale = 2)

Base plus scaled Index with 8 or 32-bit displacement	Address is Base + (2^{scale}*index)
	+displacement
		mul $t0,$s2,4
		add $t0,$t0,$s1
		lw $s0,100($t0)
		(scale = 2)

Intel 80x86 5/7

•Integer Operations

8086	support for both 8-bit and 16-bit data types (word)
80386	32-bits addresses and data (double words)

•Every operation works on both 8-bit data and on one longer data size (16 or 32-bits)

•Four major classes

- •Data movement instructions (move, push, and pop)
- •Arithmetic and Logic Instructions
- •Control Flow (conditional branches, unconditional jumps, calls, and returns)
  - •Conditional branches are based on condition codes (flags) set as a side effect of operation, most are used to compare the value of a result to zero.
    - •For argument: occur as part of normal operations and are faster to test than to compare registers in MIPS (beq and bne)
    - •Against argument: compare to zero extends operation time, and programmer has to use compare instructions to test a value that is not the result of an operation
- •String instructions (string move and compare)

Intel 80x86 6/7

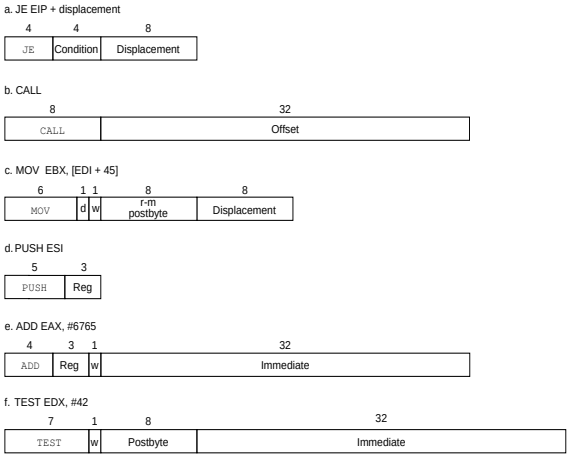
Instruction	Function
JE name	`If equal (CC) EIP = name};` `EIP – 128 ≤ name < EIP + 128`
JMP name	`{EIP = NAME};`
CALL name	`SP = SP – 4; M[SP] = EIP + 5; EIP = name;`
MOVW EBX,[EDI + 45]	`EBX = M [EDI + 45]`
PUSH ESI	`SP = SP – 4; M[SP] = ESI`
POP EDI	`EDI = M[SP]; SP = SP + 4`
ADD EAX,#6765	`EAX =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

Typical 80x86 instructions

In addition, see figure 3.33

Intel 80x86 7/7

Instruction Encoding



•Instructions vary from 1 to 17 bytes in length

•Opcode byte usually contains a bit saying whether the operand is 8 or 32 bits (1-bit w)

•1-bit d direction of move from or to memory

•For some instructions, the opcode may include the addressing mode and the register.

•Some instructions use a post-byte (extra opcode byte) which contains the addressing mode information