

---

# Complex Instruction Sets

CSE 410, Spring 2004  
Computer Systems

<http://www.cs.washington.edu/education/courses/410/04sp/>

# Readings and References

---

- Reading

- » Sections 3.12 - 3.15, *Computer Organization & Design*, Patterson and Hennessy

- Other References

- » VAX MACRO and Instruction Set Reference Manual (Compaq)
- » PowerPC Assembly Language Reference (IBM)

# CISC and RISC

---

- Complex Instruction Set Computer
  - » VAX: about 325 instructions
  - » uses very powerful instructions to do the work of the program
- Reduced Instruction Set Computer
  - » MIPS: about 100 instructions
  - » use many simple instructions to do the same amount of work faster

# Digital Equipment VAX - 1977

---

- Advances in microcode technology made complex instructions possible
- Memory was expensive
  - » so small program size = good
- Compilers had a long way to go
  - » so ease of translation from high-level language to assembly = good

# VAX Instructions

---

- Stack manipulation
  - » pop, push registers
- Queue manipulation
  - » insert, remove entries at head or tail
- Cyclic Redundancy Check
- Character Strings
  - » Compare, Locate, Match, Move, Scan, Skip, Span
  - » Move Translated Until Character

# MOVTUC

---

## Move Translated Until Character

The **source string** specified by the source length and source address operands **is translated**. It replaces the destination string specified by the destination length and destination address operands. Translation is accomplished by using **each byte of the source string as an index into a 256-byte table** whose first entry address (entry number 0) is specified by the table address operand. The **byte selected replaces the byte** of the destination string. Translation continues until a translated byte is equal to the escape byte, or until the source string or destination string is exhausted. If translation is terminated because of escape, the condition code V-bit is set; otherwise, it is cleared.

# Impact on a pipeline?

---

“During the execution of the character string instructions, pending interrupt conditions are tested. If any conditions are found, the control block is updated, a first-part-done bit is set in the processor status longword (PSL), and the instruction is interrupted. After the interruption, the instruction resumes transparently.”

# Intel 80x86 - 1978 and on

---

- 8086
  - » extension of the 8-bit 8080 microprocessor
  - » dedicated register usage requirements
- 80286, 80386
  - » segmented 32-bit address space
  - » general purpose registers
- 80486, Pentiums
  - » performance, parallelism

# Backward Compatible

---

- A key feature and a major problem is that the Intel architecture has maintained backward compatibility
  - » Good: old programs run on new CPUs
  - » Bad: new CPUs must implement the complex designs of the past
- Intel has repeatedly pushed the architecture beyond what was thought possible

# The RISC Reaction

---

- Complex instructions
  - » Take longer to execute
  - » Take more hardware to implement
- Idea: compose simple, fast instructions
  - » Less hardware is required
  - » Faster execution speed if done right
- PUSHR vs. SW + SW + SW

# Born to Pipeline - MIPS 1985

---

- Instructions all one length
  - » simplifies Instruction Fetch stage
- Regular format
  - » simplifies Instruction Decode
- Few memory operands, only registers
  - » only lw and sw instructions access memory
- Aligned memory operands
  - » only one memory access per operand

# PowerPC - 1993 and on

---

- Branch unit
  - » prefetch instructions and analyze
  - » substitute destination instruction for branch
- Superscalar
  - » multiple functional units operating in parallel
- Memory access
  - » unaligned load / store
  - » big-endian, little-endian

# Clever isn't always Good

---

“While the PowerPC architecture provides both load and store multiple instructions for GPRs, it discourages their use because their implementation on some machines may not be optimal. In fact, use of the load and store multiple instructions on some future implementations may be significantly slower than the equivalent series of single word loads or stores.”

# int strlen(char \*s)

---

```
int strlen(char *s) {  
    char *p = s;  
    while (*p != '\0') p++;  
    return p-s;  
}
```

# strlen on a Pentium

---

| <u>Address</u> | <u>Machine Code</u> | <u>Assembly Code</u>                  |
|----------------|---------------------|---------------------------------------|
| 00000          | 8b 4c 24 04         | <b>mov</b> ecx, DWORD PTR _s\$[esp-4] |
| 00004          | 8b c1               | <b>mov</b> eax, ecx                   |
| 00006          | 80 39 00            | <b>cmp</b> BYTE PTR [ecx], 0          |
| 00009          | 74 06               | <b>je</b> SHORT \$L36                 |
| \$L35:         |                     |                                       |
| 0000b          | 40                  | <b>inc</b> eax                        |
| 0000c          | 80 38 00            | <b>cmp</b> BYTE PTR [eax], 0          |
| 0000f          | 75 fa               | <b>jne</b> SHORT \$L35                |
| \$L36:         |                     |                                       |
| 00011          | 2b c1               | <b>sub</b> eax, ecx                   |
| 00013          | c3                  | <b>ret</b> 0                          |

# strlen on a PowerPC

---

| <u>Address</u> | <u>Machine Code</u> | <u>Assembly Code</u>  |
|----------------|---------------------|-----------------------|
| 00000000       | 93E1FFFC            | <b>stw</b> r31,-4(SP) |
| 00000004       | 7C7F1B78            | <b>mr</b> r31,r3      |
| 00000008       | 48000008            | <b>b</b> *+8          |
| 0000000C       | 3BFF0001            | <b>addi</b> r31,r31,1 |
| 00000010       | 889F0000            | <b>lbz</b> r4,0(r31)  |
| 00000014       | 7C840774            | <b>extsb</b> r4,r4    |
| 00000018       | 2C040000            | <b>cmpwi</b> r4,0     |
| 0000001C       | 4082FFF0            | <b>bne</b> *-16       |
| 00000020       | 7C63F850            | <b>sub</b> r3,r31,r3  |
| 00000024       | 83E1FFFC            | <b>lwz</b> r31,-4(SP) |
| 00000028       | 4E800020            | <b>blr</b>            |

# strlen on a MIPS

---

| <u>Address</u> | <u>Machine Code</u> | <u>Assembly Code</u>       |
|----------------|---------------------|----------------------------|
| 00000000       | 80820000            | <b>lb</b> \$v0,0(\$a0)     |
| 00000004       | 00041821            | <b>move</b> \$v1,\$a0      |
| 00000008       | 10400006            | <b>beqz</b> \$v0,done      |
| 0000000c       | 24630001            | <b>addu</b> \$v1,\$v1,1    |
| loop:          |                     |                            |
| 00000010       | 80620000            | <b>lb</b> \$v0,0(\$v1)     |
| 00000014       | 24630001            | <b>addu</b> \$v1,\$v1,1    |
| 00000018       | 1440ffffe           | <b>bnez</b> \$v0,loop      |
| 0000001c       | 2463fffff           | <b>subu</b> \$v1,\$v1,1    |
| done:          |                     |                            |
| 00000020       | 00641023            | <b>subu</b> \$v0,\$v1,\$a0 |
| 00000024       | 03e00008            | <b>j</b> \$ra              |