CSSE 232
Computer Architecture I

Fall 2013-2014

Other Architectures

Class Status

Reading for today

- 2.16-17

Outline

- Load-store
- Accumulator
- Memory-to-Memory
- Stack
- Advantages/Disadvantages

Load-Store architectures

- All operations occur in registers
- Register-to-register instructions have three operands per instruction
- Special instructions to access main memory

Other Architectures

- Accumulator
- Memory-to-memory
- Stack

Accumulator Architecture

- All operations use an accumulator register
- Operands come from the accumulator or memory
- Result of most operations is stored in the accumulator

Memory-to-Memory Architecture

- Similar to load-store architectures, but no registers
- All three operands of each instruction are in memory

Stack Architecture

- All operations occur on top of the stack
- Only push and pop access memory
- All other instructions remove their operands from the stack and replace them with the result
- The implementation uses a stack for the top two entries
- Accesses that use other stack positions are memory references

Examples

- Want to execute the statement
  `a = b + c;`
- `a`, `b`, and `c` are variables in memory

Accumulator

Accumulator

```
load  AddressB # Acc = Memory[AddrB] or Acc = B
add   AddressC # Acc = B + Memory[AddrC] or Acc = B + C
store AddressA # Memory[AddrA] = Acc or A = B + C
```

Memory-to-Memory

Memory-to-Memory

```
add AddressA, AddressB, AddressC
```

Stack

Stack

```
push AddressC # Top = Top+4; Stack[Top] = Memory[AddrC]
push AddressB # Top = Top+4; Stack[Top] = Memory[AddrB]
add           # Stack[Top-4] = Stack[Top] + Stack[Top-4];
              # Top = Top-4
pop AddressA  # Memory[AddrA] = Stack[Top]; Top = Top - 4
```

Load-Store

Load-Store

```
load  r1 AddressB # r1 = Memory[AddressB]
load  r2 AddressC # r2 = Memory[AddressC]
add   r3 r1 r2    # r3 = r1 + r2
store r3 AddressA # Memory[AddressA] = r3
```

Group Assignment

- What do you think are the advantages/disadvantages of each of the types of architectures?

Advantages and Disadvantages

- Stack Advantages
  - Short instructions.
- Stack Disadvantages
  - A stack can't be randomly accessed
  - Hard to generate efficient code
  - The stack itself is accessed every operation and becomes a bottleneck
- Accumulator Advantages
  - Short instructions
- Accumulator Disadvantages
  - The accumulator is only temporary storage so memory traffic is the high for this approach.

Advantages and Disadvantages

- Load-Store Advantages
  - Makes code generation easy
  - Data can be stored for long periods in registers.
- Load-Store Disadvantages
  - All operands must be named leading to longer instructions.
- Memory-to-Memory Advantages
  - Simple processor design
- Memory-to-Memory Disadvantages
  - Same as L+S
  - Bottleneck at memory access

Alternative designs

Complete Instruction Set Computer (CISC)

- Examples
  - IBM 360, DEC VAX, Intel 80x86, Motorola 68xxx
- Features
  - Varying instruction lengths
  - Memory locations as operands
  - Source operand is also the destination

Reduced Instruction Set Computer (RISC)

- Examples
  - MIPS, Alpha, PowerPC, SPARC
- Load/store
- Same instruction lengths
- Longer code programs

Hello world in x86 for Linux

```
section .data                  ; section for initialized data
str: db 'Hello world!', 0Ah ; message with new-line at the end
str_len: equ $ - str           ; calcs string length by subtracting
                               ; this' address ($) from string address

section .text                  ; this is the code section
global _start                  ; _start is the entry point
_start:                        ; procedure start
   mov    eax, 4               ; specify the sys_write function code
   mov    ebx, 1               ; specify file descriptor stdout
   mov    ecx, str             ; move string start address to ecx register
   mov    edx, str_len         ; move length of message
   int    80h                  ; tell kernel to perform the system call
   mov    eax, 1               ; specify sys_exit function code
   mov    ebx, 0               ; specify return code for OS
   int    80h                  ; tell kernel to perform system call
```

Hello world in MIPS for SPIM

```
.data
msg:      .asciiz "Hello, world!"
.text
.globl main
main:
          la $a0,msg
          li $v0,4
          syscall
          jr $ra
```

Hello world in PS2 MIPS for Linux

```
# hello.S by Spencer T. Parkin

        .rdata                          # begin read-only data segment
        .align  2                       # because of the way memory is built
hello:  .asciz  "Hello, world!\n"      # a null terminated string
        .align  4                       # because of the way memory is built
length: .word   . - hello               # length = IC - (hello-addr)

        .text                           # begin code segment
        .globl  main                    # for gcc/ld linking
        .ent    main                    # for gdb debugging info.
main:
        move    a0,$0                   # load stdout fd
        la      a1,hello                # load string address
        lw      a2,length               # load string length
        li      v0,__NR_write           # specify system write service
        syscall                         # call the kernel (write string)
        li      v0,0                    # load return code
        j       ra                      # return to caller
        .end    main                    # for dgb debugging info.
```

From `http://tldp.org/HOWTO/Assembly-HOWTO/mips.html`

Review and Questions

- Load-store
- Accumulator
- Memory-to-Memory
- Stack
- Advantages/Disadvantages

Project time

Work on `relprime()`