Announcements

- Program was due on today at 10:30 AM
- This is the final lecture
  - the next lecture will be a review for the final
- The final is Friday 8:00 - 10:00
  - it will be in this room
- Grades through part "a" of the programming assignment are on the class web page
- **There was a problem with the submit program, please login to your account and check for a mail message.**

Using more than one Processor

- Can speed up computations that take too long
- Cost is a function of volume (units shipped)
  - if you use several cheap processors instead of one expensive one you could save money
- How to connect multiple computers together
  - can use a bus or switches
    - generally buses are limited in number of processors
  - how do processors communicate?
    - send messages to each other
    - share some (or all) of their memory

Programming Parallel Computers

- Need to figure out (in software) who does what
  - called de-composing the problem
  - two major options
    - control parallel - each processor has part of the task, but can use all of the data
    - data parallel - each processor has part of the data, but performs all tasks for that data
- Programming is more difficult
  - debugging is more complex
    - several things can happen at once
  - performance counts
    - if it didn't matter, would just use one computer

What does the final cover?

- The final is cumulative, but greater emphasis will be placed on the material since the last exam
  – think of the final as two exams a third midterm and a final
- Mano
  – Chapters 1-6 (except 6-4)
  – Chapter 8 (sections 3 and 7)
  – Chapter 12 (sections 5 to 7)
- SPIM
  – MIPS assembly language
- Lecture material
  – especially the last three lectures which were not really covered in the book.

Logic and Gates

- Logic Gates
  - AND, NAND, OR, NOR, XOR, NXOR, NOT, BUFFER
  - Boolean Algebra
    - Sum of products form
    - Product of sums form
  - K-Maps
    - 2, 3, and 4 variable forms
    - can get sum of products or product of sums out
    - don’t care conditions
- Flip-flops
  - add state and time to circuits
  - characteristic tables
    - S-R, J-K ,D ,T
  - Clocks

Sequential Circuits

- Combinational logic plus flip-flops
- Representations
  - State tables
  - State Diagrams (graphs)
  - Circuit diagram
  - should be able to convert between any form
- Always have output as a function of just the current state when designing circuits

Digital Building Blocks

- Without State
  - Adder
  - Decoders (n to 2^n)
  - Encoders (2^n to n)
  - Multiplexors
- With State
  - Registers
  - Shift Registers
  - Binary Counters

- Memory
  - cells have a fixed number of bits (called a word)
    - meaning of what is stored in a word depends on situation
    - 8 bits are a byte
  - each memory word is assigned a specific address
  - There are several types of memory
    - RAM (Random Access Memory) - read/write
    - ROM (Read Only Memory)
  - Read Only memory comes in several types
- Data Representation
  - how to represent letters (ASCII)
  - representing negative numbers
    - Can use signed magnitude
    - 1's complement (flip all the bits)
    - 2's complement (flip all the bits and add one)
  - Limited space to represent numbers
    - can have overflow

ALU

- Math Functions
  - addition
  - subtraction (a-b --> a + b’ + 1)
  - use full addres for each bit position
- Shift Operations
  - logical, circular, arithmetic
- Logic operations
  - AND, OR, NOT
  - used for selective bit operations
    - set, complement, clear
- Often Use Mux to Implement an ALU

Stored Program Computer

- Can build "stored program" FSM
  - use EPROM to hold transitions
  - use register to store current state
- Stored Program Computer
  - instructions and data are stored in memory
  - instructions
    - standardized set of operations on registers and memory
    - a sequence of micro-instructions defines the instructions
    - normally execute instructions in memory order
      - unless we have a branch instruction

Instruction Format

- Instruction consists of several fields
  - op code (what to do)
  - operand (where to do it)
    - can be registers or memory
  - addressing mode
    - indirect memory references
- Fixed size of instruction limits
  - # of instructions (different bit patterns for op-codes)
  - range of memory locations (width of address field)

- Use a collection of registers to hold:

AC	Accumulator
DR	Data Register (holds memory operands)
AR	Address register (12 bits)
IR	Instruction Register
PC	Program Counter -address of instruction (12 bits)
TR	Temporary register
INPR	Input Register (8 bits)
OUTR	Output Register (8 bits)

In addition to registers and memory:

  – Have a bus to move information around
  – ALU to compute results
  – Controller to make it all work

Examples of Micro-Instructions

- AR <- IR
  - S = 101, LD_{AR}=1
- PC <- PC + 1
  - INR_{PC}=1
- DR <- AC, AC <- DR
  - S=100, LD_{DR} =1, ALU (F=A), LD_{AC}=1
- Each clock pulse is one line of RTL
- Several things can happen on one clock pulse

Executing Instructions

- List of instructions is called a program
  - instructions are stored in memory
- Computer executes next instruction in the program
  - branch instruction changes the next instruction to execute.
  - programs start execution at memory location 0.
  - one instruction at a time is executed.
- State
  - memory and registers are assumed to start empty (all 0's)
  - after executing an instruction any change to memory or registers remains.
- To keep track of steps in executing an instruction
  - use a micro-sequencer
  - really just a fancy name for a counter with clear

- Fetch and Decode steps

  T0: AR <- PC

  T1: IR <- M[AR], PC <- PC + 1

  T2: D0,….D7 <- decode(IR(12-14)),

  AR <- IR (0-11), I <- IR(15)

- Next Execution machine specific part of instruction
  – for memory reference instructions
    - D7’ I T3: AR <- M[AR]
- Subroutine calliing
  – BSA Instruction can be used to call a subroutine
  – BUN Indirect is used to return
- Interrupts
  – like a hardware subroutine

Summary of Material Since Last Exam

- Details of ALU and control logic of Mano's computer
- Assembly language
  - Mano
  - SPIM
- Modern Computer Features
  - virtual memory
  - caches
- I/O Devices
  - disks
  - networks

Assembly Language

- Provides a (slightly) higher level of abstraction
  - Programs are still written one instruction at a time.
  - provides a more symbolic way to write a program.
- Assembly Language Statement has:
  - label - symbolic names
    - for data and instructions
  - instruction field
    - Opcode (e.g. ADD)
    - operand
      - can be a label
      - can be a number
    - indirect flag (I)
  - comment fields
    - a “/”

SPIM (MIPS) Architecture

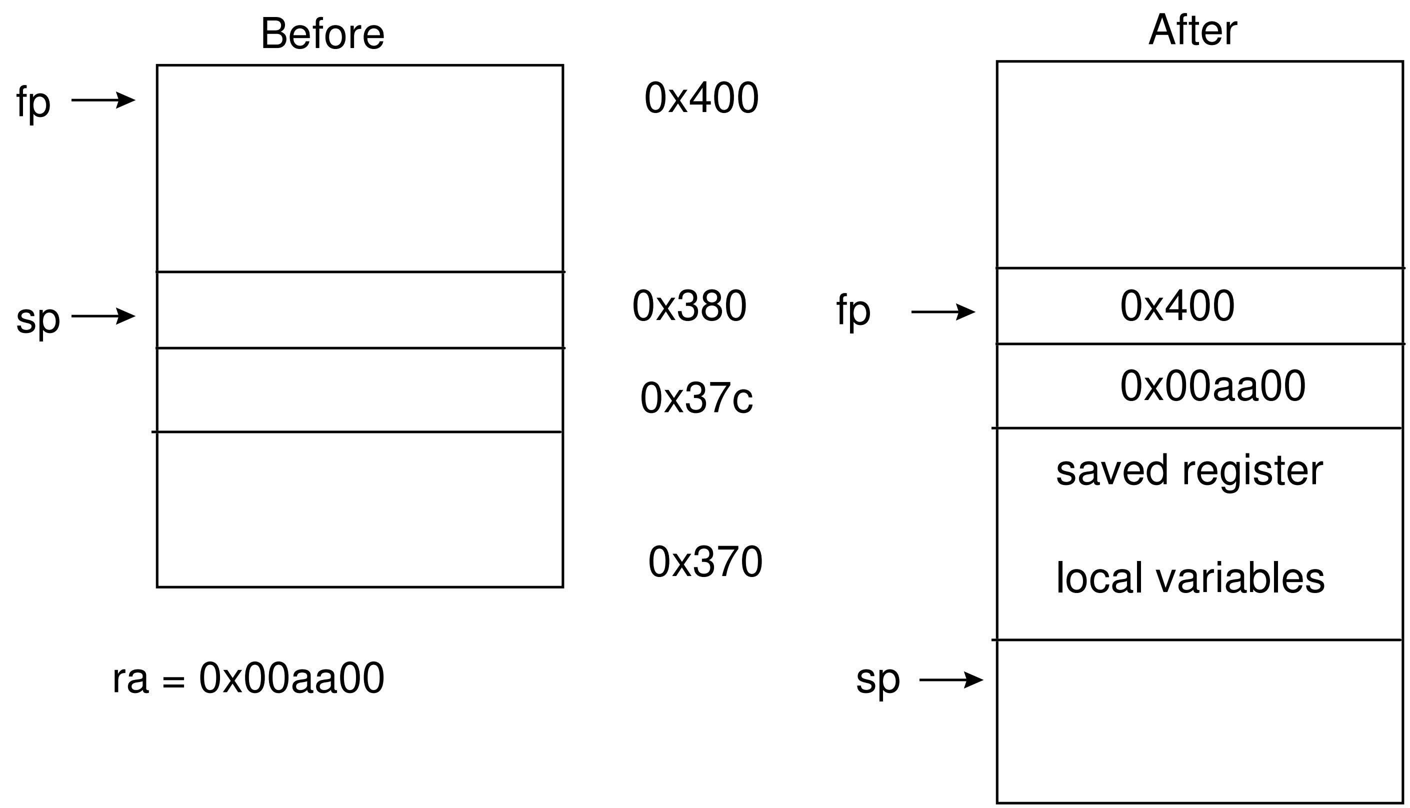
- 32 bit architecture
  - registers are 32 bits wide
  - memory addresses are 32 bits (2^{30} words each 4 bytes)
  - instructions are 32 bits wide
- Load/Store Architecture
  - can only load or store values to memory
    - I.e. there is not increment memory location.
    - no indirect addressing mode
- instructions use “three addresses”
  - destination, source1, source2
  - add t0, t1, t2
- has 32 registers
  - but one of them contains a zero

Stack Frame

- To setup stack frame:

```
sw $fp, 0($sp)
sw $ra, -4($sp)
move $fp, $sp
subu $sp, $sp, 8
```

at least 8, but could be more

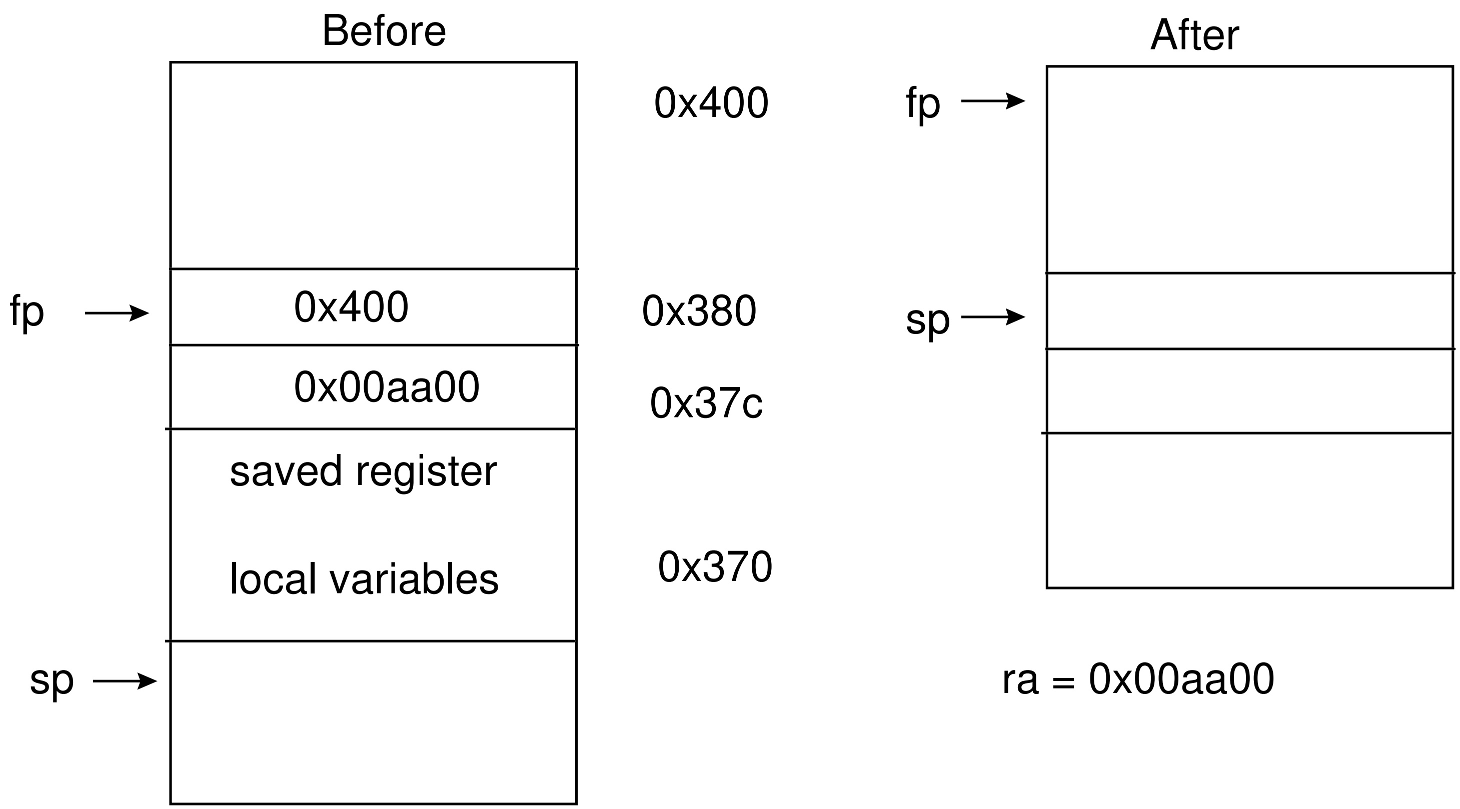


Return from Subroutine

- Cleanup stack frame:

```
lw $ra, -4($fp)
lw $fp, 0($fp)
addu $sp, $sp, 8
jr $ra
```

at least 8, but could be more



Program Control Instructions

- How do we create instructions that branch based on the relative values of two registers?
  - Answer: add status bits
- Status Bits
  - reflect the state of the last arithmetic operation
  - common status (condition) bits
    - Zero - result all zeros
    - Carry - carry out of the last bit position
    - Sign - is the sign bit set in the result?
    - V - overflow (xor of carry in and carry out of sign bit)
  - compare instruction
    - computes A - B
    - does not store result
    - updates status bits

Computer Architecture

- What is a processor architecture?
  - Organization of a processor as seen by software
    - number of size of registers
    - instructions
      - what instructions are built into hardware
    - addressing modes
      - indirection
      - auto increment modes
    - size of memory
  - It is different than the implementation
    - internal registers
    - timing of instructions
    - actual amount of memory installed/available

Instruction Formats

The format of the instruction tells us a lot about an architecture

- OpCodes
  - indicates the types of operations the machine can perform
  - complexity of instructions
- Instruction length
  - dictates available bits for op-codes and operands
  - can be fixed (MIPS, Mano) or variable length (Intel)
- Number of Registers
  - many registers (MIPS)
  - one register (Mano)
  - zero registers (stack machines)

RISC Architecture

- Reduced Instruction Set Computer
- Sometimes less is more!
- Put fewer features in hardware, but make them run very fast
- Use lots of registers
- Fixed length instructions
- Most instructions require one cycle to execute
- Computer Architecture based on quantitative studies
  - measure real programs and find out what is important

Cache Memory

- A fast place to keep frequently used information
  - caches have a fixed size that is much smaller than main memory
- If look for a value and its in the cache, it's a hit
- If look for a value and its not in the cache, it's a miss
- Key criteria for a cache is the hit ratio
  - hit ratio = $\frac{\text{\# of hits}}{\text{\# of hits + \# of misses}}$
  - possible to get a hit ratio of over 90%
  - if a hit takes 10ns and a miss 100ns, then a hit rate of 90% means an effective memory access time of 20 ns (or 5 times faster than without the cache!)

Questions about Cache Memory

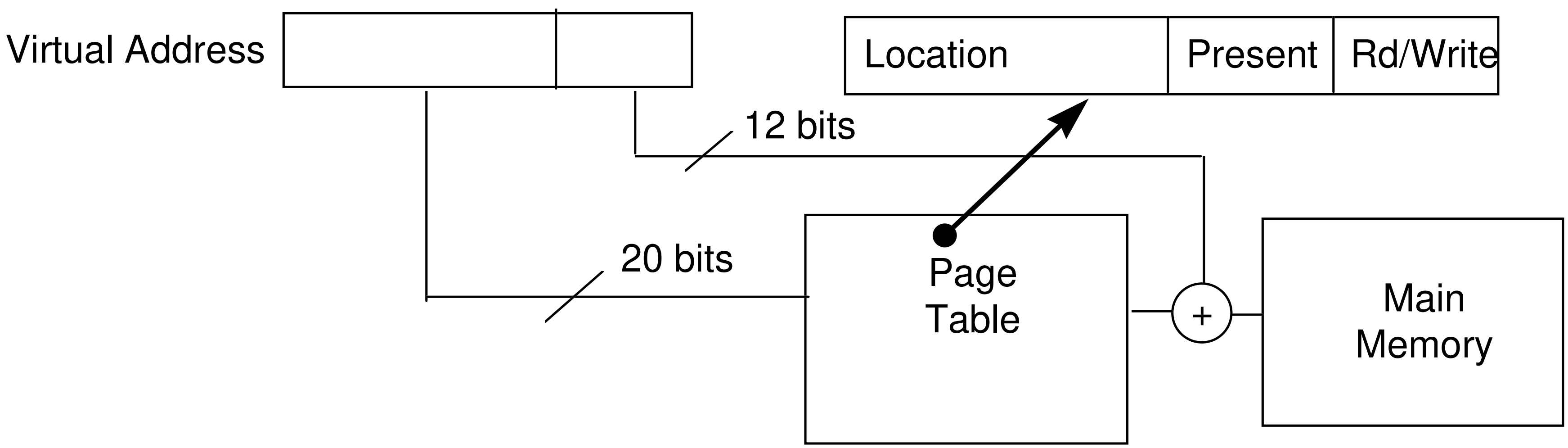
- # and size of caches
  - first level caches are typically 8 to 64k (often on the processor chip)
  - second level caches can be over a megabyte
- What to do if there is no free space in the cache?
  - need to remove something (called replacement or eviction)
  - FIFO, LRU, and random are all used
- How do we find something in the cache?
  - needs to be fast!
  - several ways to find data in a cache
- Associative mapping
  - a table with two values per item (address and value)
  - hardware looks in all locations at once
  - very fast, but expensive

Managing Memory

- Main memory is big, but what if we run out
  - keep part of memory on disk
- *virtual addresses*
  - look like normal addresses
  - hardware translates them to *physical addresses*

Virtual Addressing

- Divide physical memory into fixed sized chunks called pages
  - typical pages are 512 bytes to 64k bytes
- Have a table to map virtual pages to physical pages
- Consider a 32 bit addresses
  - 4096 byte pages (12 bits for the page)
  - 20 bits for the page number



Disks

- Several types:
  - Hard Disks - rigid surface with magnetic coating
  - Floppy disks - flexible surface with magnetic coating
  - Optical (read only, write once, multi-write)
- Hard Disk Drives:
  - collect of platters
  - platters contain concentric rings called tracks
  - tracks are divided into fixed sized units called sectors
  - a cylinder is a collection of all tracks equal distant from the center of disk
  - Current Performance:
    - capacity: megabytes to tens of gigabytes
    - throughput: sustained < 10 megabytes/sec
    - latency: mili-seconds

Networks

- Ethernet
  - 10 Mbps (to 100 Mbps)
  - mili-second latency
  - limited to several kilometers in distance
  - variable sized units of transmission
  - bus based protocol
- ATM
  - 155Mbps and up
  - fixed sized unit of transmission called a cell
    - cells are 48 bytes plus 5 bytes header
  - switch based protocol
  - for both local area and wide area networking

Making Processors Faster

- Pipelinging
  - different parts of the CPU are dedicated to parts of executing an instruction
    - one part does the fetch
    - another the decode
    - while yet a third does the memory access
  - Sort of like a washer/dryer pair
  - Each stage in the pipeline works on a different instruction
- Superscalar
  - Two separate copies of the parts of the processor
    - registers are shared
  - Each part works on a different instruction
  - Sort of like having two combined washer/dryer units