CS 316:
A Full Processor

Kavita Bala

Fall 2007

Computer Science

Cornell University

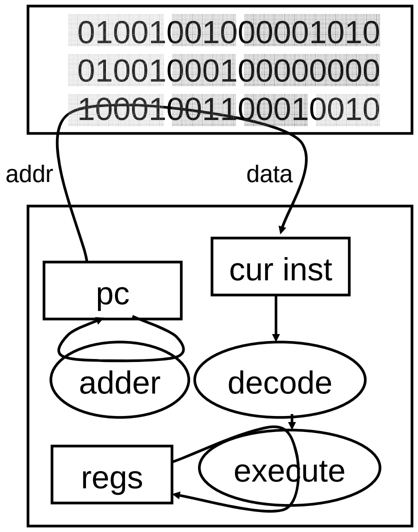
Announcements

- PA 2 is out
- Office Hours
  - Will talk about jump and delay slots
  - My office hours are canceled today, but I can meet by appointment
  - Section today can handle my office hours

Instruction Usage

- Instructions are stored in memory, encoded in binary
- A basic processor
  - fetches
  - decodes
  - executes

  one instruction at a time



Instruction Set Architecture

- The types of operations permissible in machine language define the ISA
  - MIPS: load/store, arithmetic, control flow, …
  - VAX: load/store, arithmetic, control flow, strings, …
  - Cray: vector operations, …
- Two classes of ISAs
  - Reduced Instruction Set Computers (RISC)
  - Complex Instruction Set Computers (CISC)

- We’ll study the MIPS ISA in this course

Instructions

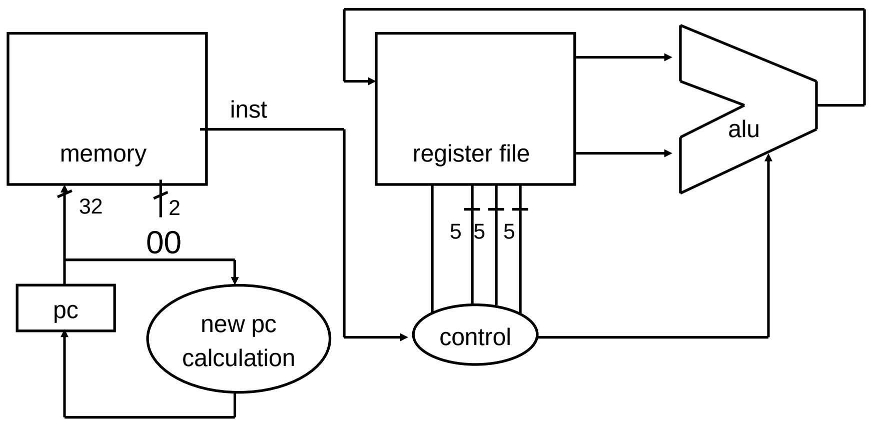
- Load/store architecture
  - Data must be in registers to be operated on
  - Keeps hardware simple
- Emphasis on efficient implementation
- Integer data types:
  - byte: 8 bits
  - half-words: 16 bits
  - words: 32 bits
- MIPS supports signed and unsigned data types

Arithmetic Instructions

op	rs	rt	rd	shamt	func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- if op == 0 && func == 0x21
  - R[rd] = R[rs] + R[rt] (unsigned)
- if op == 0 && func == 0x23
  - R[rd] = R[rs] - R[rt] (unsigned)
- if op == 0 && func == 0x25
  - R[rd] = R[rs] | R[rt]

Arithmetic Ops



Kavita Bala, Computer Science, Cornell University

Arithmetic Instructions (1)

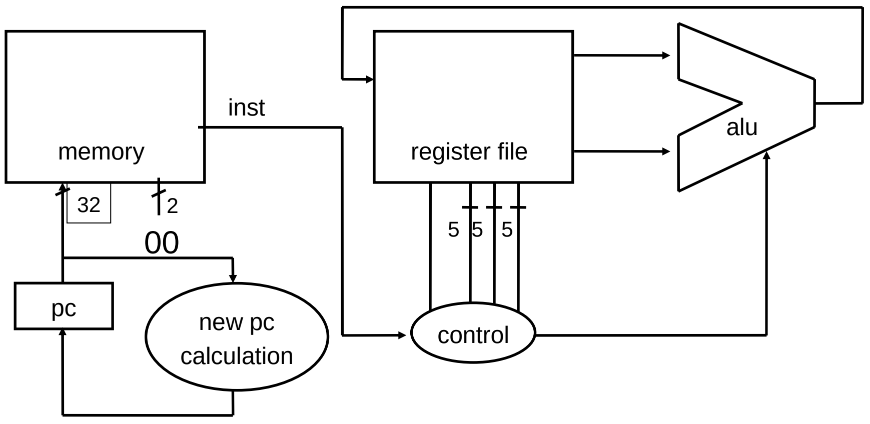
op	rs	rt	rd	shamt	func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- if op == 0 && func == 0x21
  – R[rd] = R[rs] + R[rt]
- if op == 0 && func == 0x23
  – R[rd] = R[rs] - R[rt]
- if op == 0 && func == 0x25
  – R[rd] = R[rs] | R[rt]

ADD rd, rs, rt
ADDU rd, rs, rt
AND rd, rs, rt
OR rd, rs, rt
NOR rd, rs, rt

Kavita Bala, Computer Science, Cornell University

Arithmetic Ops



Arithmetic Instructions (2)

op	rs	rd	immediate
6 bits	5 bits	5 bits	16 bits

- if op == 8
  - R[rd] = R[rs] + sign_extend(immediate)
- if op == 12
  - R[rd] = R[rs] & immediate

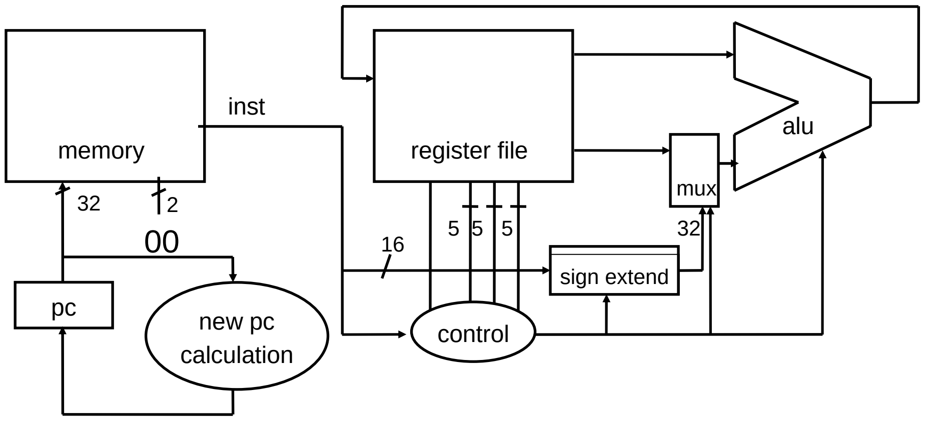
ADDI rd, rs, val
ADDIU rd, rs, val
ANDI rd, rs, val
ORI rd, rs, val

Sign Extension

- Often need to convert a small (8-bit or 16-bit) signed value to a larger (16-bit or 32-bit) signed value
  - "1" in 8 bits: 00000001
  - "1" in 16 bits: 0000000000000001
  - "-1" in 8 bits: 11111111
  - "-1" in 16 bits: 1111111111111111
- Conversion from small to larger numbers involves replicating the sign bit

Kavita Bala, Computer Science, Cornell University

Arithmetic Ops with Immediates



Kavita Bala, Computer Science, Cornell University

MIPS Instruction Types

- Arithmetic/Logical
  - three operands: result + two sources
  - operands: registers, 16-bit immediates
  - signed and unsigned versions
- Memory Access
  - load/store between registers and memory
  - half-word and byte operations
- Control flow
  - conditional branches: pc-relative addresses
  - jumps: fixed offsets

MIPS Design Principles

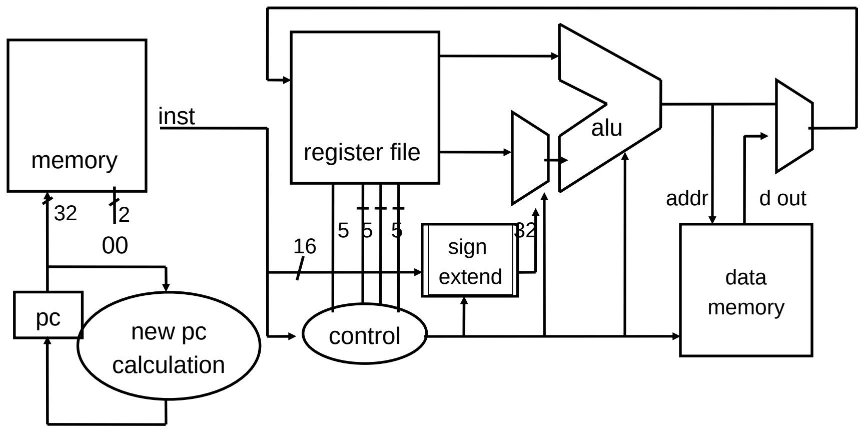
- Simplicity favors regularity
  - 32 bit instructions
- Smaller is faster
  - Small register file
- Make the common case fast
  - Include support for constants
- Good design demands good compromises
  - Support for different type of interpretations/classes

Memory Operations

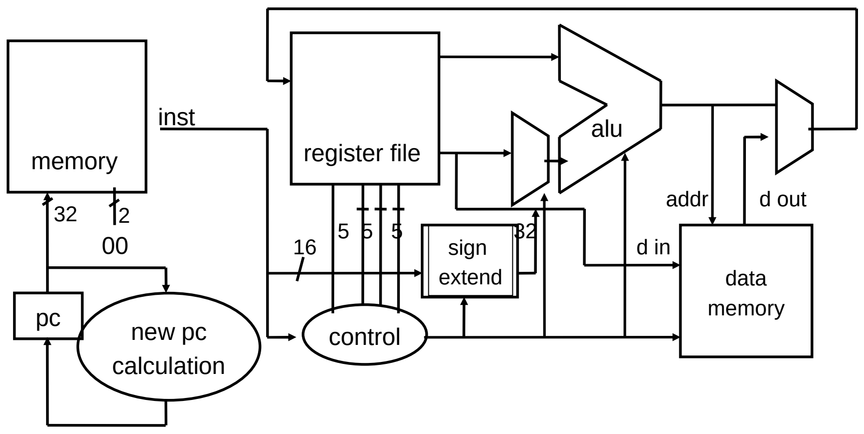
op	rs	rd	immediate
6 bits	5 bits	5 bits	16 bits

- lb, lbu, lh, lhu, lw
- sb, sh, sw
- Examples rs = r4 = addr of array; r3 = rd
  - lw r3, 0(r4)   int array[32]; x = array[0]
  - lw r3, 16(r4)   int array[32]; x = array[4]

Load



Store



Endianness

- Take a 32-bit hexadecimal value
  - e.g. 0x0A0B0C0D
- Store the word at location 0x1000
- Read the byte at location 0x1000
  - What do you get?
- Little endian

0x1003	0x1002	0x1001	0x1000
0A	0B	0C	0D

- Big endian

0x1003	0x1002	0x1001	0x1000
0D	0C	0B	0A

Endianness

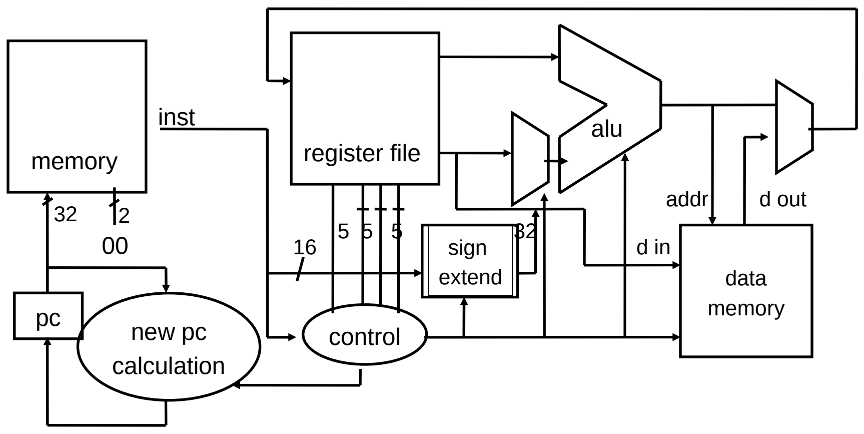
- The way the bytes are ordered within a word generally does not matter
- Except when casting between integral data types
    - casting four bytes to an int
    - casting two bytes to a short
    - casting two shorts to an int
- Such casting is typically required when sending data from one host to another
    - networks use big-endian representation for ints
    - x86's use little-endian representation for ints

Control Flow (Absolute Jump)

op	target
6 bits	26 bits

- j, jal
- Absolute addressing
- new PC = high 4 bits of current PC || target || 00
    - Cannot jump from 0xffff000000000000 to 0x0000100000000000
    - Better to make all instructions fit in 32 bits than to support really large absolute jumps
- Examples
    - j L01 goto L01

Absolute Jump

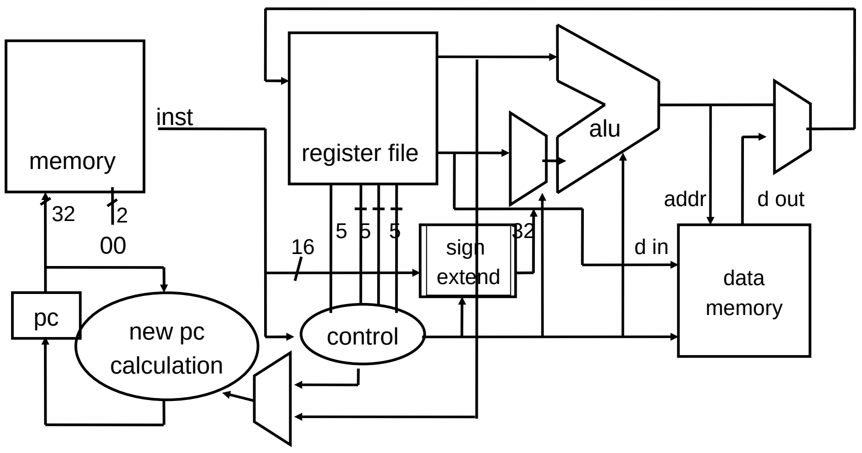


Control Flow (Jump Register)

op	rs	000 000 000 000 000	func
6 bits	5 bits	15 bits	6 bits

- new PC = R[rs]
- Can jump to any address stored in a register

Jump Register



Kavita Bala, Computer Science, Cornell University

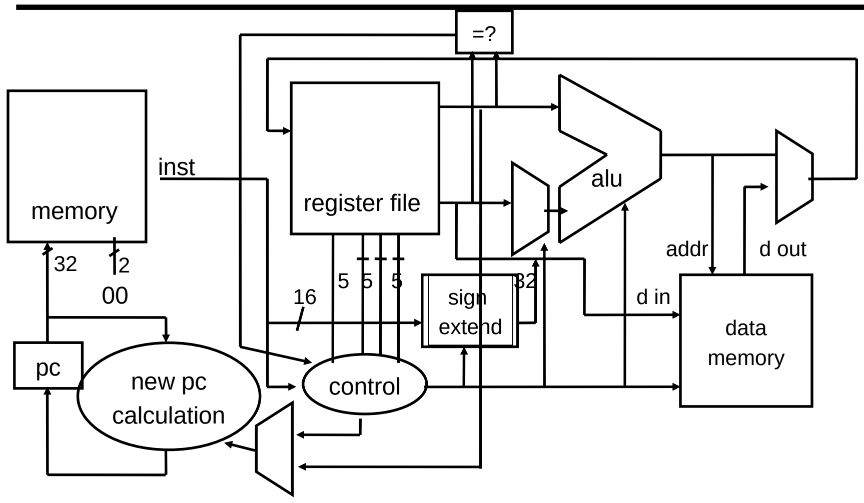
Control Flow (Branches)

op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

- Some branches depend on the relative values of two registers
- if op == 4   # BEQ
  - if R[rs] == R[rt]
    - new PC = old PC + sign_extend(immediate << 2)
- BEQ, BNE

Kavita Bala, Computer Science, Cornell University

Branch

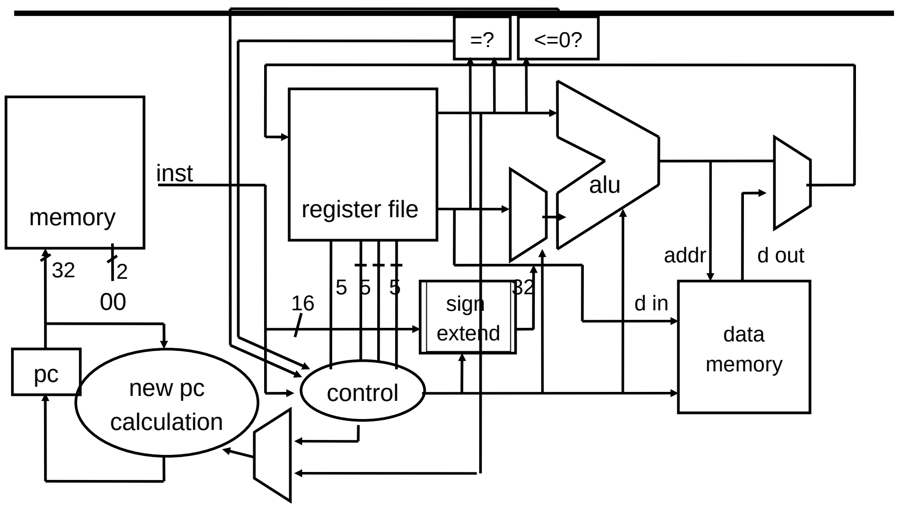


Control Flow (Branches)

op	rs	subop	immediate
6 bits	5 bits	5 bits	16 bits

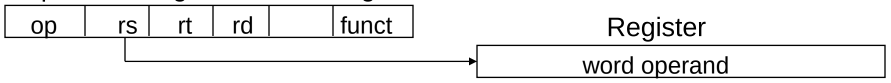
- Some branches depend on the value of a single register
- if op == 1 && subop == BLTZ
  - if R[rs] < 0
    - new PC = old PC + sign_extend(immediate << 2)
- BGEZ, BGTZ, BLTZ, BLEZ

Branch

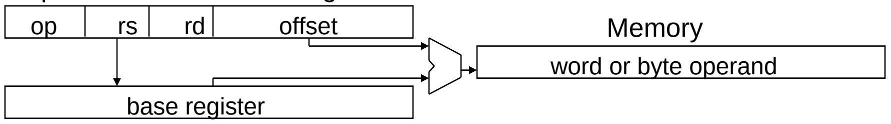


MIPS Addressing Modes

1. Operand: Register addressing

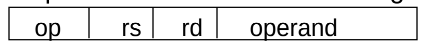


2. Operand: Base addressing

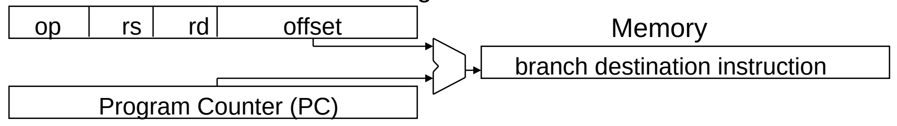


MIPS Addressing Modes

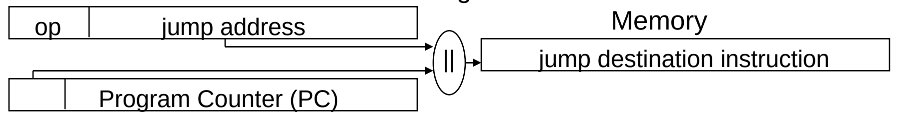
3. Operand: Immediate addressing



4. Instruction: PC-relative addressing



5. Instruction: Pseudo-direct addressing



Examples

- A[12] = h + A[8]

- lw, $t0, 32($s3)
- add $t0, $s2, $t0
- sw $t0, 48($s3)

Example

- if (i == j) f = g + h else f = g – h

- bne $s3, $s4, Else
- add $s0, $s1, $s2
- j Exit
- nop
- Else: sub $s0, $s1, $s2
- nop
- Exit:

Summary

- Full-blown processor