CS 316:
Memory and Processor

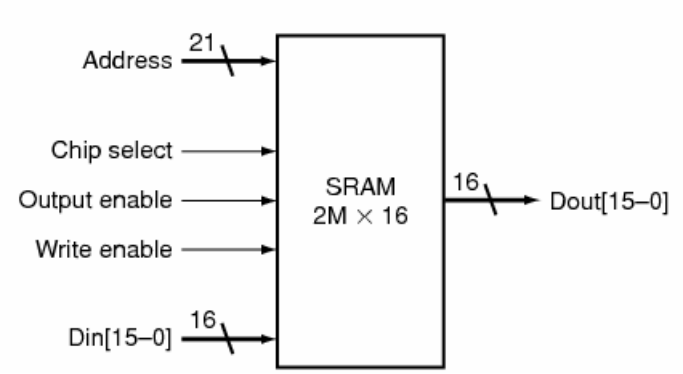
Kavita Bala
Fall 2007
Computer Science
Cornell University

Memory

- Various technologies
  - S-RAM, D-RAM, NV-RAM
- Non-Volatile RAM
  - Data remains valid even through power outages
  - More expensive
  - Limited lifetime; after 100000 to 1M writes, NV-RAM degrades
  - Flash cards

Static RAM: SRAM

- Static-RAM
  - So called because once stored, data values are stable as long as electricity is supplied
  - Based on regular flip-flops with gates



How to build large memories?

- Cannot use a 2M->1 multiplexer!
- Use a shared line (called bit line)
- Multiple memory cells can assert line
  - Need 3 state buffer
  - 3 states: asserted (1), deasserted (0), or high impedance

Big Memories

- Tri state buffer got rid of big mux
- But still need a big decoder to pick the right entry
  - 4M x 8 SRAM requires
    - 22 to 4M decoder
    - And 4M lines!
- Instead
  - Rectangular arrays
  - 2-step decode

Parallel Memory Banks

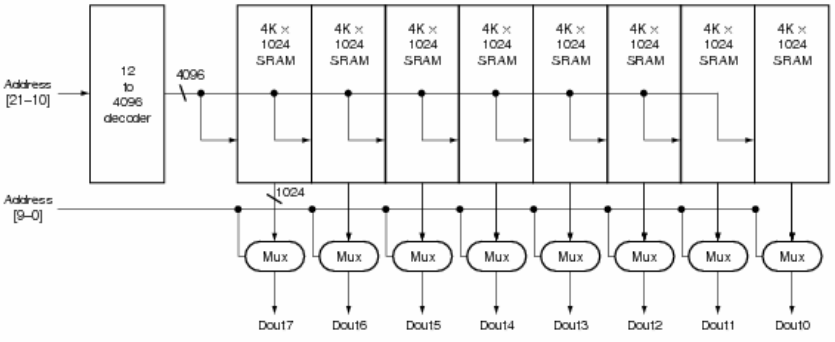


FIGURE B.9.4 Typical organization of a 4M x 8 SRAM as an array of 4K x 1024 arrays. The first decoder generates the addresses for eight 4K × 1024 arrays; then a set of multiplexors is used to select 1 bit from each 1024-bit-wide array. This is a much easier design than a single-level decode that would need either an enormous decoder or a gigantic multiplexor. In practice, a modern SRAM of this size would probably use an even larger number of blocks, each somewhat smaller.

SRAM

- Needs a few gates per cell
- Used for caches (we talk about this later)
- For higher density, use DRAM

Dynamic RAM: DRAM

- Dynamic-RAM
  - Data values require constant refresh
  - Internal circuitry keeps capacitor charges

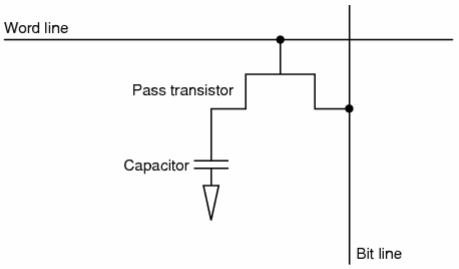


FIGURE B.9.5 A single-transistor DRAM cell contains a capacitor that stores the cell contents and a transistor used to access the cell.

DRAM

- Single transistor vs. many gates
  - Denser and cheaper
- But, need refresh
  - Read and write back
  - Every few milliseconds…
  - Also organized in 2D grid, so can do rows at a time
  - Done independently on chip
- Hence, slower

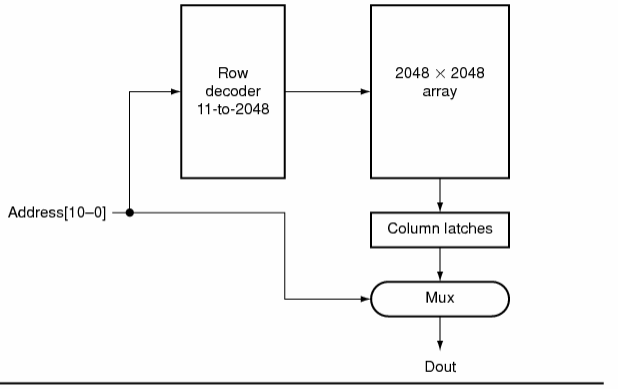


FIGURE B.9.6 A 4M × 1 DRAM is built with a 2048 × 2048 array. The row access uses 11 bits to select a row, which is then latched in 2048 1-bit latches. A multiplexor chooses the output bit from these 2048 latches. The RAS and CAS signals control whether the address lines are sent to the row decoder or column multiplexor.

Summary

- We now have enough building blocks to build machines that can perform non-trivial computational tasks

- SRAM: caches
- DRAM: main memory

A Simple Processor

Instructions

```
for(i = 0; I < 10; ++i)
    printf("go cornell cs");
```

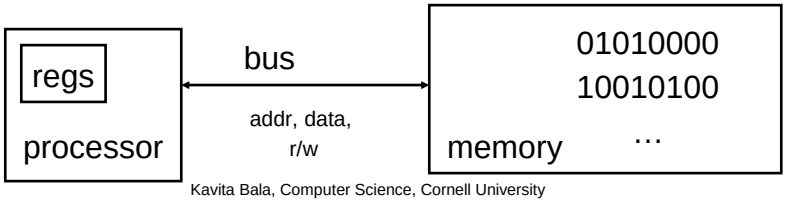
```
li r2, 10
li r1, 0
slt r3, r1, r2
bne …
```

```
01001001000001010
01001000100000000
10001001100010010
```

- Programs are written in a high-level language
  - C, Java, Python, Miranda, …
  - Loops, control flow, variables
- Need translation to a lower-level computer understandable format
  - Processors operate on machine language
  - Assembler is human-readable machine language

Basic Computer System

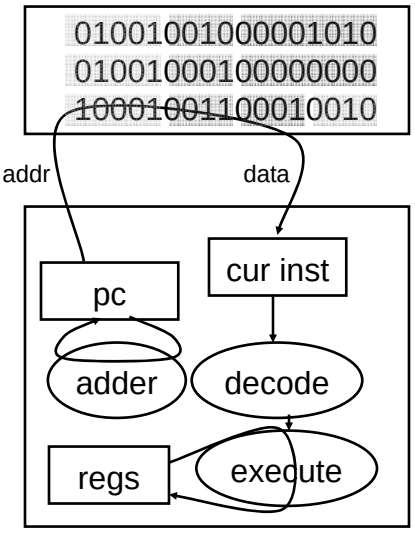
- A processor executes instructions
  - Processor has some internal state in storage elements (registers)
- A memory holds instructions and data
  - Harvard architecture: separate insts and data
  - von Neumann architecture: combined inst and data
- A bus connects the two



Instruction Usage

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

  one instruction at a time



Instruction Types

- Arithmetic
  - add, subtract, shift left, shift right, multiply, divide
  - compare
- Control flow
  - unconditional jumps
  - conditional jumps (branches)
  - subroutine call and return
- Memory
  - load value from memory to a register
  - store value to memory from a register
- Many other instructions are possible
  - vector add/sub/mul/div, string operations, store internal state of processor, restore internal state of processor, manipulate coprocessor

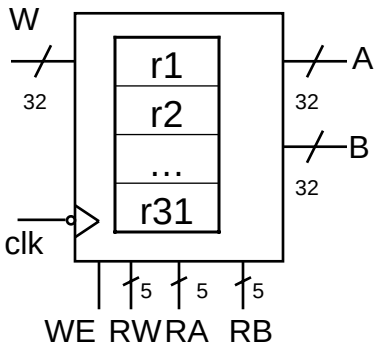
Instruction Set Architecture

- 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

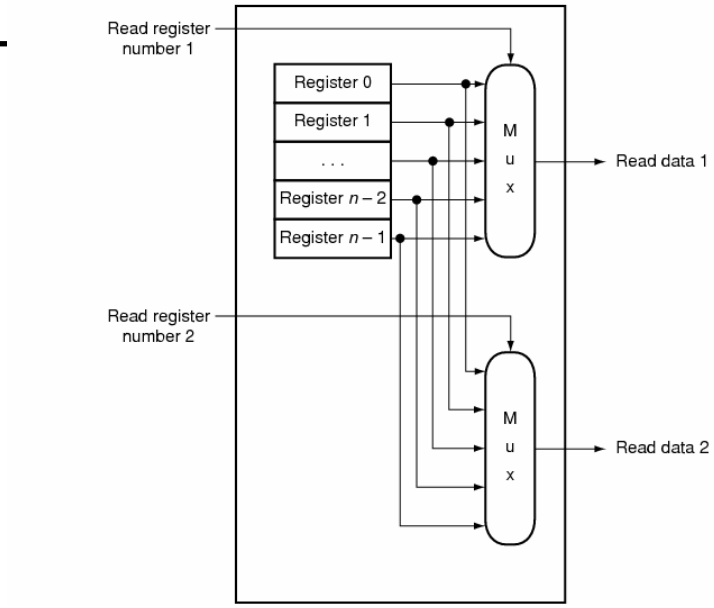
Register file



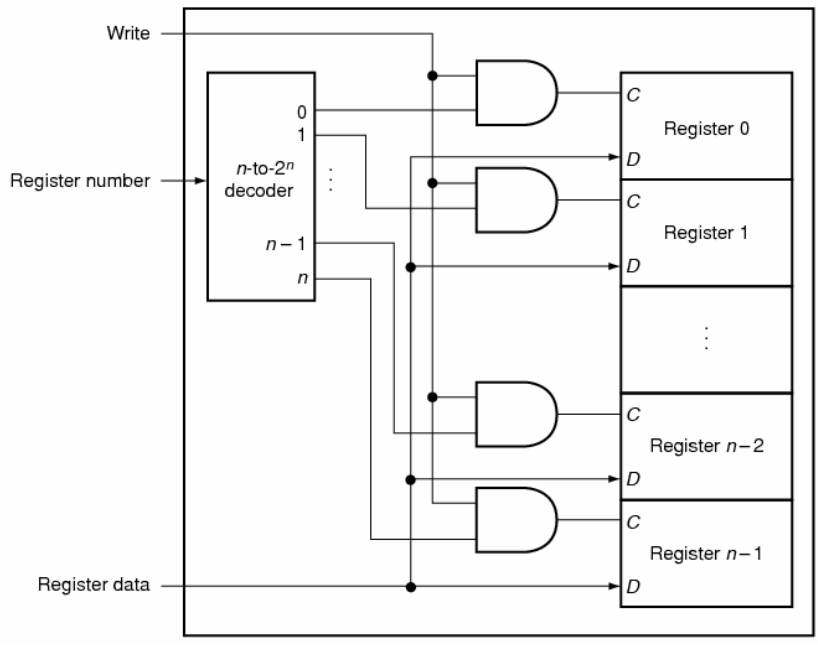
- The MIPS register file
  - 32 32-bit registers
  - register 0 is permanently wired to 0
  - Write-Enable and RW determine which reg to modify
  - Two output ports A and B
  - RA and RB choose values read on outputs A and B
  - Reads are combinatorial
  - Writes occur on falling edge if WE is high

Register File

- Set of registers
  - Read or written
  - Use register number to access it
- Read or write ports
  - Decoder for each port
- D flip flops to store bits

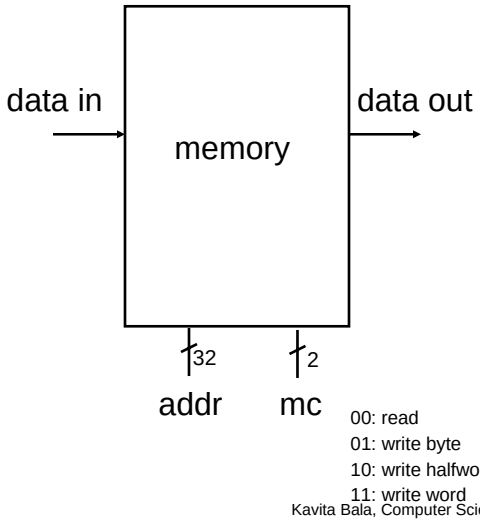


Kavita Bala, Computer Science, Cornell University



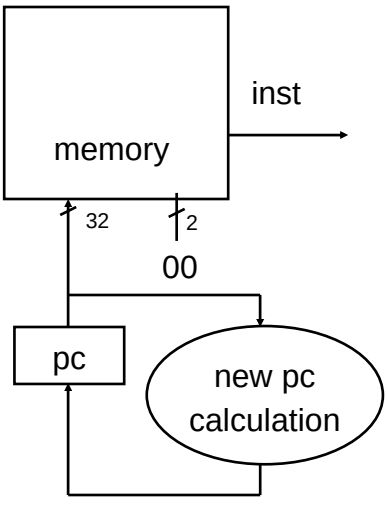
Kavita Bala, Computer Science, Cornell University

Memory



- 32-bit address
- 32-bit data
  - word = 32 bits
- 2-bit memory control input

Instruction Fetch



- Read instruction from memory
- Calculate address of next instruction
- Fetch next instruction

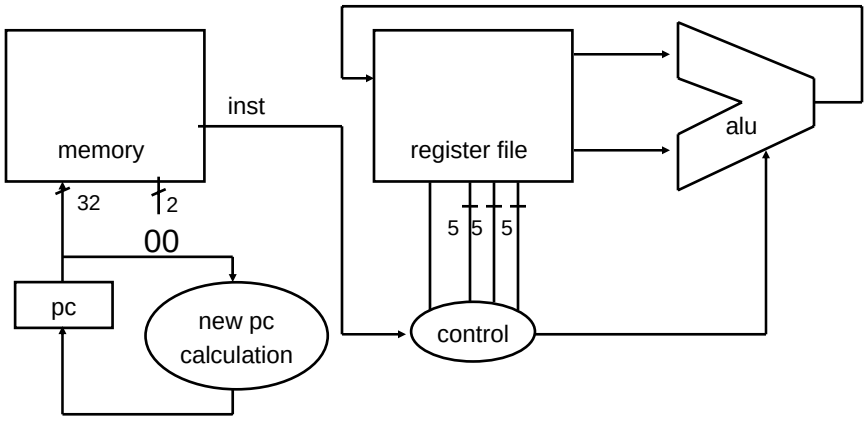
MIPS Design Principles

- Simplicity favors regularity
- Smaller is faster
- Make the common case fast
- Good design demands good compromises

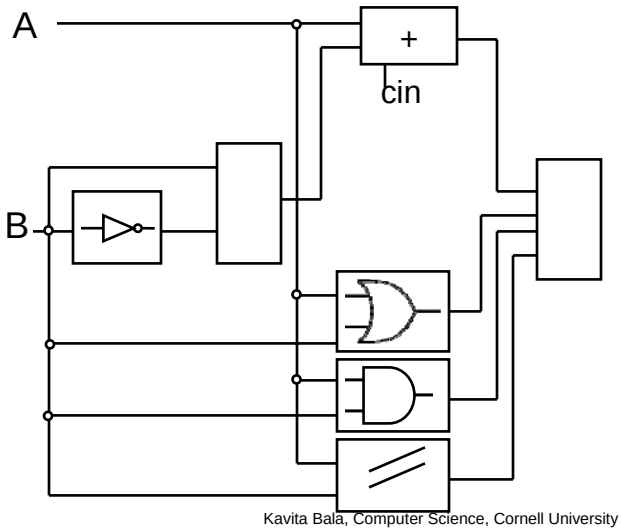
Arithmetic Instructions

- 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

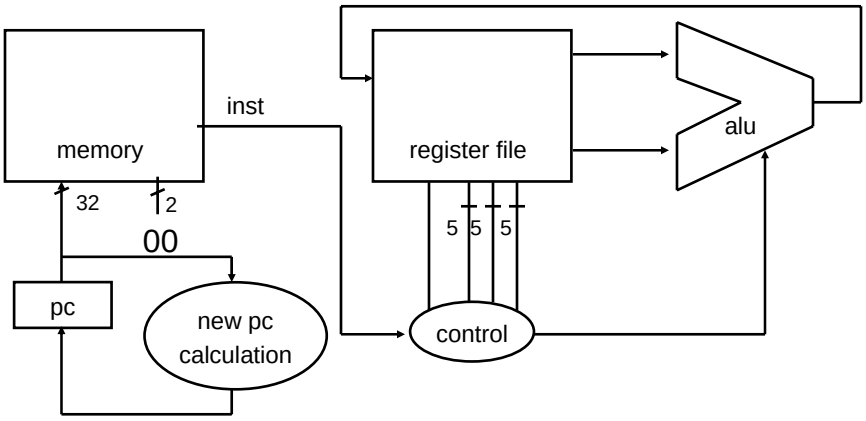


Arithmetic Logic Unit



- Implements add, sub, or, and, shift-left, …
- Operates on operands in parallel
- Control mux selects desired output

Arithmetic Ops



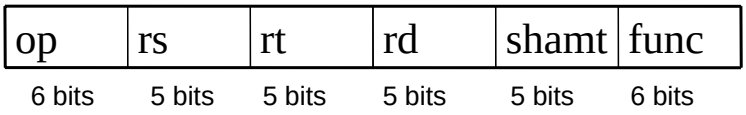
Summary

- With an ALU and fetch-decode unit, we have the equivalent of Babbage's computation engine
  - Much faster, more reliable, no mechanical parts
- Next: control flow and memory operations

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

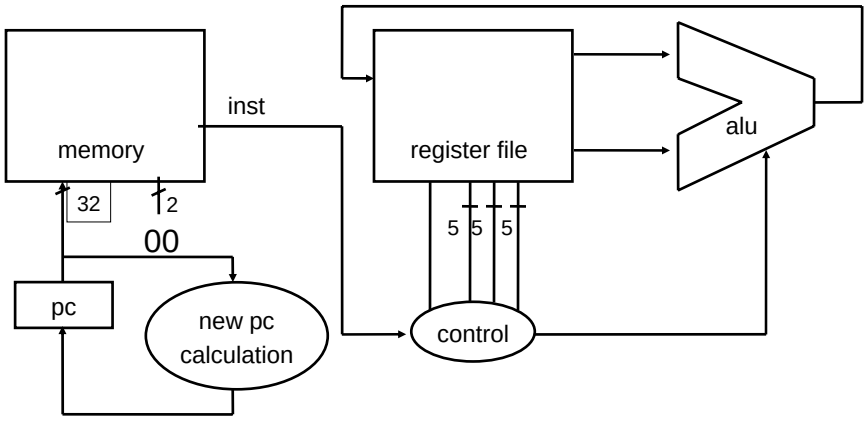
Arithmetic Instructions (1)



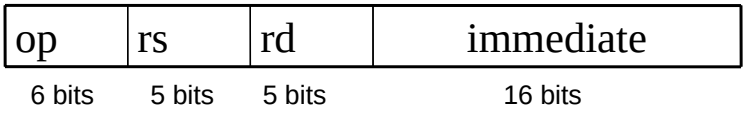
- if op == 0 && func == 0x21
  - R[rd] = R[rs] + R[rt]
- if op == 0 && func == 0x23
  - R[rd] = R[rs] - R[rt]
- if op == 0 && func == 0x25
  - R[rd] = R[rs] | R[rt]

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

Arithmetic Ops



Arithmetic Instructions (2)



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

ADDI rs, rt, val
ADDIU rs, rt, val
ANDI rs, rt, val
ORI rs, rt, 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

Arithmetic Ops with Immediates