CSE140: Components and Design Techniques for Digital Systems

Memory

Tajana Simunic Rosing
Welcome to CSE 140!

• **Where we are going today:**
  – Non-volatile memory & CPUs

• **Upcoming:**
  – HW7 due Thursday
  – Midterm #3 in CENTR 101 on 3/18 at 11:30am

• **CAPEs are out!!!** [https://cape.ucsd.edu/students/](https://cape.ucsd.edu/students/)
  – Your feedback is very important to us! We read your feedback carefully and use it for future courses. 😊
  – **At 90% CAPE response rate, I will drop the lowest HW grade!**

• **Bonus points for the class:**
  – Bonus take home final exam, posted on Monday, 3/20, due Saturday, 3/25
  – Worth 2% of the course grade

• **Regrades:**
  – Grades posted Th, regrade due the following Th; updated grades posted on TED by the weekend
  – Check your grades in TED, let us know if anything is missing

• **Discussion session** on Friday at 2pm!
Non-volatile memory yesterday and today

- **Erasable Programmable ROM (EPROM)**
  - Uses “floating-gate transistor” in each cell
  - Programmer uses higher-than-normal voltage so electrons *tunnel* into the gate
    - Electrons become trapped in the gate
    - Only done for cells that should store 0
    - Other cells will be 1
  - To erase, shine ultraviolet light onto chip
    - Gives trapped electrons energy to escape
    - Requires chip package to have window

- **Electronically-Erasable Programmable ROM (EEPROM)**
  - Programming similar to EPROM
  - Erasing one word at a time *electronically*

- **Flash memory**
  - Like EEPROM, but large blocks of words can be erased *simultaneously*

- **EEPROM & FLASH are in-system programmable**
Non-volatile memory going forward

• A new class of data storage/memory devices
• Emerging NVMs have exciting features:
  – Non-volatile like Flash (~ 10 years)
  – Fast access times (~ SRAM)
  – High density (~ DRAM)
• NVM *blurs the distinction* between
  – Memory (*fast, expensive, volatile*) &
  – Storage (*slow, cheap, non-volatile*)
• Key issues:
  – Slow writes, low endurance, costly and complex manufacturing

Sources: TSR, Cummings, KFR
1T-1C FeRAM

- Similar in construction to DRAM
  - Both cell types include one capacitor and one access transistor
  - DRAM cell capacitor use a linear dielectric; FeRAM includes ferroelectric material, typically lead zirconate titanate (PZT)
  - Writing is accomplished by applying a field across the ferroelectric layer by charging the plates on either side of it, forcing the atoms inside into the "up"/logic “1” or "down"/logic "0"

- Advantage:
  - No need for refresh – 99% lower power than DRAM
  - Similar performance to DRAM

- Disadvantage:
  - It is unclear how it will scale as materials stop being ferroelectric at small sizes (now produced in 130nm)
  - Reads are destructive – data has to be rewritten
STT-RAM: Spin-Transfer Torque RAM

- The spin torque direction of electrons to flip a bit in a magnetic tunneling junction (MTJ)

- **Advantages:**
  - High endurance & fast reads

- **Disadvantages:**
  - *Write energy:* large current needed to reorient the magnetization for most commercial applications;
  - *Asymmetric write:* Writing a “1” needs much more time and energy than writing a “0”
Domain Wall Memory (DWM)

- Similar to STT-RAM structure
- **Advantage:**
  - needs only one tunneling barrier and fixed layer → area savings
- **Disadvantages:**
  - complexity of design, read/write delay due to sequential access
Shift-based DWM

- Writes by shifting data of one of the two fixed layers with the desirable direction comp
- **Advantage**: faster writes than a traditional DWM
- **Disadvantage**: cost and manufacturing complexity

1-bit DWM is fast

Multi-bit DWM is area efficient, but needs extra latency for shifting

Sources: TSR, Cummings, KFR
**PCM: Phase Change Memory**

**Phase Change Memory (PCM)**
- Flips a bit by changing the state of material
- Crystalline (SET) and amorphous (RESET) phase

**Advantages:**
- better scalability than other emerging technologies

**Disadvantages:**
- Slow writes
- Low endurance ($10^7$ writes)

Candidate for DRAM replacement
ReRAM: Resistive RAM

- Two types: Access-based (1T-1R) and crossbar ReRAM (1T-nR)

- **Access-based ReRAM (1T-1R)**
  - A dielectric, which is normally an insulator, can conduct with sufficiently high voltage

- **Advantage:**
  - Very fast reads and writes ~ 20ns
  - Very high density

- **Disadvantage:**
  - Limited endurance (10^5 writes)
Crossbar ReRAM

- Crossbar ReRAM (1T-nR)
- Advantage:
  - Highly scalable
  - Can be implemented at the top of the chip with in 3D architecture
  - Very low energy consumption
  - Low cost (possible replacement for Flash)
- Disadvantage:
  - Much slower than 1T-1R ~us

Sources: TSR, Cummings, KFR
Crossbar RRAM Applications

**Crossbar 1T-nR**

- Data Center – Block Oriented Data Storage
  - Low latency ($10^{-5}$ s), high bandwidth, low energy
  - Native 3D compatible array architecture
  - Alternatives: SSD, DRAM (disk cache), hard disk

**1T-1R**

- Nodes – XiP Code and Random Access Data
  - Low latency ($10^{-8}$ s), high bandwidth, low energy
  - Alternatives: embedded NOR, discrete NOR

NVRAM Comparison

The diagram compares various memory technologies based on their power-off data storage and high speed operation capabilities. The axes represent VDDmin (V) on the x-axis and access time (ns) on the y-axis. Key technologies include SRAM, ROM, DRAM, MRAM, PCM, ReRAM, Flash, and Emerging NVM.
### NVMs Comparison

- **STT-RAM**: SRAM cache replacement
- **PCRAM**: DRAM main memory and storage
- **ReRAM**: NAND flash, embedded NOR flash

<table>
<thead>
<tr>
<th>Features</th>
<th>SRAM</th>
<th>eDRAM</th>
<th>STT-RAM</th>
<th>PCRAM</th>
<th>ReRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density Speed</td>
<td>Low</td>
<td>High</td>
<td>High (Fast for read; slow for write)</td>
<td>Very high (Slow for read; very slow for write)</td>
<td>Very high (Slow for read/write)</td>
</tr>
<tr>
<td>Dynamic Power</td>
<td>Low</td>
<td>Medium</td>
<td>Low for read; very high for write</td>
<td>Medium for read; high for write</td>
<td>Medium for read; high for write</td>
</tr>
<tr>
<td>Leakage Power</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Non-volatility</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Sources: TSR, Cummings, KFR
Summary

• Memory hierarchy
  – Needs: speed, low power, predictable

• Cache design
  – Mapping, replacement & write policies

• Memory types
  – ROM vs RAM vs NVM

• NVM
  – Many new technologies that are still maturing
  – Excellent target for big data and energy-efficient applications
CSE140: Components and Design Techniques for Digital Systems

CPU design

Tajana Simunic Rosing
CPU Components

- **Combinational logic:**
  - Boolean equations, logic gates, logic minimization
  - Multiplexors, demux, encoders, decoders
  - ALU design

- **Sequential logic:**
  - Storage elements: SRAM, Latches, FFs, Registers
  - Counters
  - Control implemented via FSMs
The following is true about this circuit:
A. None of the below
B. This is a 2:1 mux
C. The most efficient way to implement it is with pass gates
D. It connects one of the two single bit inputs to the output
E. Both B & C above
The following is true for this circuit:
A. None of the below
B. It is a demux
C. It is a decoder
D. Its truth table is:
<table>
<thead>
<tr>
<th>I1</th>
<th>I0</th>
<th>O3</th>
<th>O2</th>
<th>O1</th>
<th>O0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
E. Both C & D
**Full 32-bit ALU**

**OP CODE**

**CarryIn**

**Result**

**Overflow**

**CarryOut**

**A** 32

**B** 32

**Performs:**
AND, OR, NOT,
ADD, SUB,
Overflow
Detection, GTE
MSB ALU

Binvert

A31

B31

CarryIn

OP

0

1

2

3

4

result

GTEin = 0

If GTEout = 1, A ≥ B

GTEout

overflow

CarryOut

xor

 xor

¬

−

+
Mystery Circuit 3

The following is true:
A. This circuit has memory
B. Input A has reset functionality
C. It is possible for A and B to be both logic 1
D. This is the truth table for this circuit:
E. All of the above

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
The following is true for this circuit:
A. It has two D-FFs in it
B. Master D-latch output is Q2
C. Inverter makes it a falling edge D-FF
D. Both B & C
E. None of the above
Read/Write Register File

Clock

Write Reg # (5 bits)

Write Value

Read Value

Read Reg # (5 bits)

Sources: TSR, Cummings, KFR
Comparing Processor Memory

- **Register file**
  - Intermediate data storage within CPU
  - Fastest (ns)
  - Biggest area/cell

- **SRAM**
  - Faster (~10ns)
  - More compact
  - Used for caches: L1 32KB, L2 2MB, L3 45MB

- **DRAM**
  - Slowest (50ns) but very compact (8GB)
    - refreshing takes time
  - Different technology due to large caps
  - Used for main memory
Static RAM (SRAM) - writing

- "Static" RAM cell writing
  - word enable comes from decoder
  - When 0, value \( d \) is stored in inverters
  - When 1, the data bit value enters
    - Example shows a "1" being written into cell
**Static RAM (SRAM) - reading**

- “Static” RAM cell - reading
  - When \(\text{rw}\) set to read, the RAM logic sets both \(\text{data}\) and \(\text{data}'\) to 1
  - The stored bit \(d\) will pull either the left line or the right bit down slightly below 1
  - “Sense amplifiers” detect which side is slightly pulled down
**Dynamic RAM (DRAM)**

- “Dynamic” RAM cell
  - 1 transistor (rather than 6)
  - Relies on *large* capacitor to store bit
    - Write: transistor conducts, data voltage level gets stored on top plate of capacitor
    - Read: look at the value of $d$
    - Problem: Capacitor discharges over time
      - Must “refresh” regularly, by reading $d$ and then writing it right back

---

Sources: TSR, Cummings, KFR
ROM & Non-volatile memory

- **Erasable Programmable ROM (EPROM)**
  - Uses “floating-gate transistor” in each cell
  - Programmer uses higher-than-normal voltage so electrons *tunnel* into the gate
    - Electrons become trapped in the gate
    - Only done for cells that should store 0, rest are 1
  - To erase, shine ultraviolet light onto chip
- **Electronically-Erasable Programmable ROM (EEPROM)**
  - Programming similar to EPROM
  - Erasing one word at a time *electronically*
- **Flash memory**
  - Large blocks can be erased *simultaneously*
- **Non-volatile memory (NVM):**
  - **Phase-change memory (PCM)**
    - Material changes phase (liquid to solid) to program
  - **STT-RAM & MRAM**
    - Uses magnetic properties to program
  - Similar to RAM, but with slower writes
CPU Control and Datapath Execute Instruction Set

Control takes program as input; it interprets each instruction and tells the Datapath to operate on data via ALU, memory and registers.
CPU Components – Single Cycle Execution

Assumptions:
- Every machine language instruction happens in 1 Clock Cycle
- MIPS architecture
  - Microprocessor without interlocked pipeline stages
  - reg-reg architecture: all operands must be in registers (total 24)
  - 3 Instruction Types; each instruction 32 bits long
    1. R-type: all data in registers (most arithmetic and logical)
      e.g. add $s1, $s2, $s3
    2. I-type: branches, memory transfers, constants
      e.g. beq $s1, $s2, Label; lw $s1, 32($s2)
    3. J-type: jumps and calls
      e.g. j Label;

```
add $s0, $s1, $s2
```

<table>
<thead>
<tr>
<th>000000</th>
<th>10001</th>
<th>10010</th>
<th>10000</th>
<th>00000</th>
<th>100000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>17</td>
<td>18</td>
<td>16</td>
<td>0</td>
<td>32</td>
</tr>
</tbody>
</table>
R-type Instruction: reg-reg ALU ops (e.g. add, and)

**OPCODE** = 0

<table>
<thead>
<tr>
<th>31-26</th>
<th>25-21</th>
<th>20-16</th>
<th>15-11</th>
<th>10-6</th>
<th>5-0</th>
</tr>
</thead>
</table>

Source Register 1 (attached to “Read Register 1” input)

Source Register 2 (attached to “Read Register 2” input)

Destination Register (attached to “Write Register” input)

**ADD $S1, $S2, $S3**

**ADD RD, RS, RT**

Tells operation to be performed

Tells specific variant of operation (e.g. add/sub have same opcode)

Shift amount (for sll, srl etc.)
Step 1 (R-type): Fetch instruction and advance PC
Step 2 (R-type): Read two registers and set control signals
Step 3 (R-type): Perform the ALU operation
Step 4 (R-type): Write result to register
I-Type: Store Instruction

<table>
<thead>
<tr>
<th>OPCODE = 35 or 43</th>
<th>RS</th>
<th>RT</th>
<th>OFFSET</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-26</td>
<td>25-21</td>
<td>20-16</td>
<td>15-0</td>
</tr>
</tbody>
</table>

- **Opcode**: Tells operation to be performed
- **RS**: Base Address Register (attached to “Read Register 1” input)
- **RT**: Source register whose value will be stored to memory (attached to “Read Register 2” input)
- **OFFSET**: Constant offset (added to the base address in RS)

**Examples**:

- `SW $S1, 32($S2)`
- `SW RT, #(RS)`

**Note**: same as x86

`MOV [ebx+32], eax`
Step 1 (store): Fetch instruction and advance PC
Step 2 (store): Read register values and set control signals
Step 3 (store): Compute the address
Step 4 (store): Write the value to memory
I-Type: Conditional Branch

<table>
<thead>
<tr>
<th>OPCODE = 4 or 5</th>
<th>RS</th>
<th>RT</th>
<th>BRANCH TARGET’S OFFSET</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-26</td>
<td>25-21</td>
<td>20-16</td>
<td>15-0</td>
</tr>
</tbody>
</table>

BEQ/BNE Instruction

Source Register 1 (attached to “Read Register 1” input)

Source register 2 (attached to “Read Register 2” input)

Word Offset, which we multiply by 4 (via <<2) to get Bit Offset, then add to PC+4 to get the address of the instruction to which we branch if RS = RT)

“PC-relative address”

BEQ Source1, Source2, Offset

BEQ $S1, $S2, 100 = AL

4 17 18 25 = ML (in binary)
Step 1 (beq): Fetch instruction and advance PC
Step 2 (beq): Read register values and set control signals
Step 3 (beq): Compare registers, calculate branch target, and choose new PC
### J-Type: Unconditional Branch

<table>
<thead>
<tr>
<th>JMP/JAL Instruction</th>
<th>OPCODE = 2 or 3</th>
<th>BRANCH TARGET ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>31-26</td>
<td>25-0</td>
</tr>
</tbody>
</table>

- **J Offset**
  - Actual Address (in words) which we multiply by 4 (<<2) to get 28-Bit Address, then concatenate to upper 4 bits of PC+4 to get the 32-bit address of instruction to which we branch unconditionally.
- **J 10000 = AL**
- **2 2500 = ML (in binary)**
Single-Cycle Datapath with Support for the Jump Instruction