CSE140: Components and Design Techniques for Digital Systems

Register Transfer Level (RTL) Design
based on Vahid chap. 5

Tajana Simunic Rosing
Where we are now

• What we’ve covered so far:
  – Chap 1, 2, 3, 5
  – Started Chap 5 from Vahid

• Bonus points:
  – HW#7 – CLA and RTL problems are for bonus points, the rest graded as normal

• Logistics:
  – HW #8 assigned, due Monday 6/10/13 at 8pm via email or in person to Sheila/TAs/Prof. Solutions will be posted right after 8pm.
  – The lowest grade of the HWs will be dropped

• CAPEs: [http://www.cape.ucsd.edu/](http://www.cape.ucsd.edu/). The site closes on Monday, 6/10, at 8am
  – If participation rate is > 85%, 2nd lowest grade of HWs will be dropped

• Where we are going next:
  – RTL: Chap 5 from Vahid’s textbook
  – Sample CPU design and review for the last exam

• The last exam is on Tue, June 11th, 3-4:20pm, same location as the lectures
  – Prof office hr June 11th 1-2:30pm
  – TAs office hrs June 10th 6-8pm Rajib, Ketan
  – June 9th 7-8pm Jingwei, June 8th 11-12pm Ryan, June 7th, 5-6pm Josh
  – No TA office hrs from June 11th onwards
Reaction timer

- On reset ($rst$) reaction timer waits for 10 sec before turning on light ($len=1$)
- Measures the length of time $rtime$ (ms) until user presses button $B$
  - If reaction slower than 2sec, output $slow=1$ and $rtime=2000$
Fast sum of 16 32-bit registers
Hot water detector

- Output warning when average temp over the past 4 samples exceeds a user defined value; clr disables the system
- Inputs (32 bit): CT – current temp; WT – warning temp
- Output: W – high if hot temperature; stays on until clr pressed again
CSE140: Components and Design Techniques for Digital Systems

Single Cycle CPU Design

Tajana Simunic Rosing
CPU Components

• Combinational logic:
  – Boolean equations, logic gates
  – Multiplexors and decoders
  – ALU: executes arithmetic/logical operations
2-input, 32-bit MUX

- Selects one input as the output

implementation
Decoder

- 2 input, $2^2 = 4$ outputs

### Implementation

Translates input into binary number B and turns on output B
Full 32-bit ALU

OP CODE

Performs:
AND, OR, NOT,
ADD, SUB,
Overflow
Detection, GTE

CarryIn

Result

Overflow

CarryOut

A
32

B
32

32

32-bit ALU
MSB ALU

A31
Binvert

B31

result

GTEin = 0

If GTEout = 1, A ≥ B

OP

0

1

2

3

4

GTEout

overflow

CarryOut

CarryIn

ADD

¬

¬

¬
ALU Design Example

<table>
<thead>
<tr>
<th>S2</th>
<th>S1</th>
<th>S0</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>B-A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>A-B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A+B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A xor B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A or B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>A and B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
CPU Components

- **Combinational logic:**
  - Boolean equations, logic gates
  - Multiplexors and decoders
  - ALU: executes arithmetic/logical operations

- **Sequential logic:**
  - Storage (memory) elements
  - Counters
Memory elements: D-Latch

- Sets SR-latch (Q) to value of D when clock (C) is high; otherwise last Q retained

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>Reset</th>
<th>Set</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>

Stores 0
Stores 1

R: reset
S: set

Stored state value

Sources: TSR, Cummings, KFR
Memory elements: Flip-Flop

- Stores new value of D in Q when C falls, otherwise current stored value of Q is retained: *falling edge-triggered clocking methodology*

\[ C \quad D \quad Q \quad \neg Q \]

\[ C1 \quad Q1 \quad D1 \quad \neg Q1 \]

\[ C2 \quad Q2 \quad D2 \quad \neg Q2 \]

\[ Q \quad \neg Q \]
Read/Write Register File

- Input Read Reg #. MUX selects Q for that set of FFs as output.
- Input Write Reg # and Value. Write Value goes to each FF. Write Reg # turns on C to only 1 FF, where Value is stored.
CPU Components

- **Combinational logic:**
  - Boolean equations, logic gates
  - Multiplexors and decoders
  - ALU: executes arithmetic/logical operations

- **Sequential logic:**
  - The clock
  - Storage (memory) elements
  - Counters

- **Datapath and Control:** logic block that executes machine language instructions
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. J-type: jumps and calls
      e.g. j Label;
    3. I-type: branches, memory transfers, constants
      e.g. beq $s1, $s2, Label; lw $s1, 32($s2)

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

```
000000 10001 10010 10000 00000 100000
0   17   18   16    0    32
```
• When an assembly language program is run:
  • Is assembled, linked, loaded into instruction memory
  • PC initialized to the address of the first instruction
    • the PC is really a 32-bit register
    • “counting” is done by separate adders
  • rest of clock cycle used to fetch/execute I, update PC
R-type Instruction: reg-reg ALU ops (e.g. add, and)

- **R-type Instruction**
  - **OPCODE**: 0
  - **RS**: 31-26
  - **RT**: 25-21
  - **RD**: 20-16
  - **shamt**: 15-11
  - **FUNCT**: 10-6
  - **Shift amount** (for sll, srl etc.)
  - **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)
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

Tells operation to be performed

<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>

Base Address Register (attached to “Read Register 1” input)

Source register whose value will be stored to memory (attached to “Read Register 2” input)

Constant offset (added to the base address in RS)

**Store Instruction**

- \( \text{SW } \$S1, 32(\$S2) \)
- \( \text{SW } \ RT, #(RS) \)

**Note:** same as \( x86 \)

\( \text{MOV } [\text{ebx+32}], \text{ eax} \)
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)

**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**

**J 10000 = AL**

Actual Address (in words) which we multiply by 4 ($\ll 2$) 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.
Single-Cycle Datapath with Support for the Jump Instruction