Review Problem 19

- What is done for these ops during the CPU’s execute steps at right?
  - ADD X0, X1, X2  STUR X3, [X4, #16]  LDUR X5, [X6, #8]

Get inst

Must be the same

Split into fields, determine the OP

Get X1, X2 from reg

Get X4 + X3, reg file #16

Compute Add

Get X6, reg file #8

Compute Addr

Read memory

Add in ALU

Store X3 to mem

Store X2 to X0 in Reg file

PC += 4

Store to X5

PC += 4

CBZ  Next Instr:  if (cond)

PC = PC + Imm

else  PC += 4

Execute

Operand Fetch

Instruction Decode

Instruction Fetch

Result Store

Next Instruction
RTL & Processor Design

Convert instructions to Register Transfer Level (RTL) specification

\[ \text{RegA} = \text{RegB} + \text{RegC}; \]

RTL specifies required interconnection of units, control

Math unit example:

1. (add): \( A = A + B; \ l++; \)
2. (mult): \( A = A * B; \ l++; \)
3. (hold): \( A = A; \ l++; \)
4. (init): \( A = \text{Din}; \ l++; \)
Instruction Fetch

\[ \text{Instruction} = \text{MEM}[\text{PC}] \]
\[ \text{PC} = \text{PC} + 4 \]
Add/Subtract RTL

Add instruction: ADD Rd, Rn, Rm

\[
\text{Instruction} = \text{Mem}[PC]; \\
\text{Reg}[Rd] = \text{Reg}[Rn] + \text{Reg}[Rm]; \\
\text{PC} = \text{PC} + 4;
\]

Subtract instruction: SUB Rd, Rn, Rm

\[
\text{Instruction} = \text{Mem}[PC]; \\
\text{Reg}[Rd] = \text{Reg}[Rn] - \text{Reg}[Rm]; \\
\text{PC} = \text{PC} + 4;
\]
Add/Subtract Datapath

\[ \text{Reg}[\text{Rd}] = \text{Reg}[\text{Rm}] \oplus \text{op} \cdot \text{Reg}[\text{Rn}] \]
Load RTL

Load Instruction: \texttt{LDUR Rd, [Rn, DAddr9]}

\begin{align*}
\text{Instruction} &= \text{Mem} \left( PC \right) \\
\text{Address} &= \text{Reg}[\text{Rn}] + \text{Sign Extend} (\text{DAddr9}) \\
\text{Reg}[\text{Rd}] &= \text{Mem}[\text{Address}] \\
\text{PC} &= \text{PC} + 4
\end{align*}
Datapath + Load

\[ \text{Addr} = \text{Reg}[Rn] + 5 \times \text{SE}(Daddr 9); \]
\[ \text{Reg}[Rd] = \text{mem}[\text{Addr}]; \]
Store RTL

Store Instruction: STUR Rd, [Rn, DAddr9]

Instruction = Mem[PC];
Addr = Reg[Rn] + SE(DAddr9);
mem[Addr] = Reg[Rd];

PC = PC + 4;
Datapath + Store

\[ \text{Addr} = \text{Reg} \left[ R_{n_j} \right] + SE \left[ \text{Daddr-9} \right] \]

Next \[ \text{Addr} [j] = \text{Reg} [R_{d_j}] \]
Review Problem 21

- Immediate vals for some instructions are sign-extended, while others are not. Build a 16bit to 64bit sign-extend unit that can handle both.
Branch RTL

Branch Instruction: B BrAddr26

Instruction = Mem [PC + J]

PC = PC + SignExt (BrAddr26) << 2

< two zeroes tacked on to the bottom.>
Datapath + Branch

\[ PC = PC + SE \ (B_{-}Add_{-}26) \ll 2 \]
Conditional Branch RTL

Conditional Branch Instruction: CBZ Rd, CondAddr19

Instruction = Mem [PC] /j

Cond = (Reg[Rd] == 0); // Use the ALU
if (Cond)
    PC = PC + SEC (CondAddr19) << 2 /j
else
    PC = PC + 4 /j
Datapath + Conditional Branch

Cond = (Reg[RegT] == 0)
if (cond)
   pc = pc + SE (Cond Addr19) << 2;
else
   pc = pc + 4;

Cond Addr19

PC

BrAddr26

<<2

Adder

4

BrTaken

Reg2Loc

Rd Rm Rn

0 1

RegWrite

Aw Ab Aa Da Dw
RegFile Db

WrEn

DAddr9

SE

ALUSrc

ALUOp

MemWrite

MemToReg

WrEn Addr Din Dout
Data Memory

opcode + cond signal
Control

Identify control points for pieces of datapath
  Instruction Fetch Unit
  ALU
  Memories
  Datapath muxes
  Etc.

Use RTL for determine per-instruction control assignments
Complete Datapath

Comb. Logic → Control

opcode

Reg2Loc

Rd Rm Rn

0 1

RegWrite

DAddr9

RegFile

Aw Ab Aa Da Dw

WrEn Db

MemToReg

Zero

MemWrite

WrEn Addr

Data Memory

Din Dout

ALUOp

ALUSrc

SE

PC

CondAddr19

SE BrAddr26

UncondBr

BrTaken

Adder

Adder
# Control Signals

<table>
<thead>
<tr>
<th>Opcode[31:26]</th>
<th>100010</th>
<th>110010</th>
<th>111110</th>
<th>111110</th>
<th>000101</th>
<th>101101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode[25:21]</td>
<td>11000</td>
<td>11000</td>
<td>00010</td>
<td>00000</td>
<td>xxxxx</td>
<td>00xxx</td>
</tr>
<tr>
<td></td>
<td>ADD</td>
<td>SUB</td>
<td>LDUR</td>
<td>STUR</td>
<td>B</td>
<td>CBZ</td>
</tr>
<tr>
<td><strong>Reg2Loc</strong></td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td><strong>ALUSrc</strong></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td><strong>MemToReg</strong></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>RegWrite</strong></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>MemWrite</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>BrTaken</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(zero)</td>
</tr>
<tr>
<td><strong>UncBr</strong></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>ALL Op</strong></td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>X</td>
<td>B=0?</td>
</tr>
</tbody>
</table>
ADD Control

Instruction = Mem[PC];
Reg[Rd] = Reg[Rn] + Reg[Rm];
PC = PC + 4;
SUB Control

Instruction = Mem[PC];
Reg[Rd] = Reg[Rn] - Reg[Rm];
PC = PC + 4;
LDUR Control

Instruction = Mem[PC];
Addr = Reg[Rn] + SignExtend(DAddr9);
Reg[Rd] = Mem[Addr];
PC = PC + 4;
Instruction = Mem[PC];
Addr = Reg[Rn] + SignExtend(DAddr9);
Mem[Addr] = Reg[Rd];
PC = PC + 4;
B Control

Instruction = Mem[PC];
PC = PC + SignExtend(BrAddr26) << 2;
CBZ Control

Instruction = Mem[PC];
Cond = (Reg[Rd] == 0);
if (Cond)
  PC = PC + SE(CondAddr19) << 2;
else
  PC = PC + 4;