Qutrits Bring Quantum Computers Closer:
An Australian group has built and tested logic gates that convert qubits into qutrits (three-level quantum states)!

But who cares: new iPhones soon? Ben hopes so...

www.slashdot.org
Review: A Single Cycle Datapath

- We have everything except control signals

---

We have everything except control signals.
Recap: Meaning of the Control Signals

- nPC_sel:
  - “n” = next
  - “+4” 0 ⇒ PC ← PC + 4
  - “br” 1 ⇒ PC ← PC + 4 + {SignExt(Im16), 00}

- Later in lecture: higher-level connection between mux and branch condition
Recap: Meaning of the Control Signals

- **ExtOp**: “zero”, “sign”
  - MemWr: 1 ⇒ write memory

- **ALUsrc**: 0 ⇒ regB;
  - MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem
  1 ⇒ immed
  - RegDst: 0 ⇒ “rt”; 1 ⇒ “rd”

- **ALUctr**: “ADD”, “SUB”, “OR”
  - RegWr: 1 ⇒ write register
### RTL: The Add Instruction

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>21</th>
<th>16</th>
<th>11</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>shamt</td>
<td>funct</td>
<td></td>
</tr>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>6 bits</td>
<td></td>
</tr>
</tbody>
</table>

- **add rd, rs, rt**
  - **MEM[PC]**: Fetch the instruction from memory
  - **R[rd] = R[rs] + R[rt]**: The actual operation
  - **PC = PC + 4**: Calculate the next instruction’s address
Instruction Fetch Unit at the Beginning of Add

- Fetch the instruction from Instruction memory: Instruction = MEM[PC]
  - same for all instructions
The Single Cycle Datapath during Add

\[ R[rd] = R[rs] + R[rt] \]
Instruction Fetch Unit at the End of Add

- \( \text{PC} = \text{PC} + 4 \)
- This is the same for all instructions except: Branch and Jump
Single Cycle Datapath during Or Immediate?

- \( R[rt] = R[rs] \ OR \ ZeroExt[Imm16] \)

![Diagram showing single cycle datapath](image)

- **Instruction**
- **ALU**
- **RegFile**
- **Data Memory**
- **Extender**

### Table: Single Cycle Datapath

<table>
<thead>
<tr>
<th></th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Circuit Diagram:

1. **RegFile**
2. **ALU**
3. **Data Memory**
4. **Extender**
5. **Instruction**
6. **ALUctr**
7. **MemtoReg**
8. **MemWr**
9. **RegDst**
10. **RegWr**
11. **nPC_sel**
Single Cycle Datapath during Or Immediate?

- \( R[rt] = R[rs] \text{ OR } \text{ZeroExt}[\text{Imm16}] \)

\[
\begin{array}{c|c|c|c|c}
\text{op} & \text{rs} & \text{rt} & \text{immediate} \\
\hline
& & & \\
\end{array}
\]

\( nPC_{sel} = +4 \)
\( \text{RegDst} = 0 \)
\( \text{RegWr} = 1 \)

\( \text{MEM} \rightarrow \text{REG} \)
The Single Cycle Datapath during Load?

- \( R[rt] = \text{Data Memory}\{R[rs] + \text{SignExt[imm16]}\} \)
The Single Cycle Datapath during Load

- \( R[rt] = \text{Data Memory \{R[rs] + \text{Sign Ext}[\text{imm16}]\} } \)

![Diagram](image)
The Single Cycle Datapath during Store?

- Data Memory \( \{ R[rs] + \text{SignExt}[\text{imm16}] \} = R[rt] \)
The Single Cycle Datapath during Store

- Data Memory \( \{R[rs] + \text{SignExt}[\text{imm16}]\} = R[rt] \)
The Single Cycle Datapath during Branch?

- if \((R[rs] - R[rt] == 0)\) then \(Zero = 1\); else \(Zero = 0\)

\[
\begin{array}{c|c|c|c|c}
\text{op} & \text{rs} & \text{rt} & \text{immediate} \\
\hline
\end{array}
\]
The Single Cycle Datapath during Branch

- if $(R[rs] - R[rt] == 0)$ then Zero = 1; else Zero = 0
Instruction Fetch Unit at the End of Branch

- if \((\text{Zero} == 1)\) then \(\text{PC} = \text{PC} + 4 + \text{SignExt}[\text{imm16}]*4\); else \(\text{PC} = \text{PC} + 4\)

**What is encoding of \(nPC\_sel\)?**
- Direct MUX select?
- Branch inst. / not branch

**Let’s pick 2nd option**

**Q: What logic gate?**
Adminstrivia

• Sorry about Proj1 woes

• Grading is rough stuff. Don’t blame Ben, he’s innocent.

• HW6 Due imminently! Students have claimed it takes a very long time
  • Remember MODULAR DESIGN. This could save you a lot of time.

• HW7 Out Now! Get started soon.

• Proj3 is on its way, will be out soon after the weekend.
Step 4: Given Datapath: RTL → Control

Instruction<31:0>

Inst Memory

Adr

Op Fun Rt Rs Rd Imm16

nPC_sel RegWr RegDst ExtOp ALUSrc ALUctr MemWr MemtoReg

DATA PATH

Control
## A Summary of the Control Signals (1/2)

<table>
<thead>
<tr>
<th>inst</th>
<th>Register Transfer</th>
</tr>
</thead>
</table>
| **add** | R[rd] ← R[rs] + R[rt]; PC ← PC + 4  
ALUsrc = RegB, ALUctr = “ADD”, RegDst = rd, RegWr, nPC_sel = “+4” |
| **sub** | R[rd] ← R[rs] – R[rt]; PC ← PC + 4  
ALUsrc = RegB, ALUctr = “SUB”, RegDst = rd, RegWr, nPC_sel = “+4” |
| **ori** | R[rt] ← R[rs] + zero_ext(Imm16); PC ← PC + 4  
ALUsrc = Im, Extop = “Z”, ALUctr = “OR”, RegDst = rt, RegWr, nPC_sel = “+4” |
| **lw** | R[rt] ← MEM[ R[rs] + sign_ext(Imm16)]; PC ← PC + 4  
ALUsrc = Im, Extop = “sn”, ALUctr = “ADD”, MemtoReg, RegDst = rt, RegWr, nPC_sel = “+4” |
| **sw** | MEM[ R[rs] + sign_ext(Imm16)] ← R[rs]; PC ← PC + 4  
ALUsrc = Im, Extop = “sn”, ALUctr = “ADD”, MemWr, nPC_sel = “+4” |
| **beq** | if ( R[rs] == R[rt] ) then PC ← PC + sign_ext(Imm16)] || 00 else PC ← PC + 4  
nPC_sel = “br”, ALUctr = “SUB” |
### A Summary of the Control Signals (2/2)

#### See Appendix A

<table>
<thead>
<tr>
<th>func</th>
<th>10 0000</th>
<th>10 0010</th>
<th>We Don’t Care :-)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00 0000</td>
<td>00 0000</td>
<td>00 1101</td>
</tr>
<tr>
<td></td>
<td>add</td>
<td>sub</td>
<td>ori</td>
</tr>
<tr>
<td>RegDst</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ALUSrc</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nPCsel</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Jump</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ExtOp</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>ALUctr&lt;2:0&gt;</td>
<td>Add</td>
<td>Subtract</td>
<td>Or</td>
</tr>
</tbody>
</table>

#### R-type

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>21</th>
<th>16</th>
<th>11</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>shamt</td>
<td>funct</td>
<td>add, sub</td>
</tr>
</tbody>
</table>

#### I-type

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>21</th>
<th>16</th>
<th>11</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td></td>
<td></td>
<td>immediate</td>
<td>ori, lw, sw, beq</td>
</tr>
</tbody>
</table>

#### J-type

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>21</th>
<th>16</th>
<th>11</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>target address</td>
</tr>
</tbody>
</table>

We Don’t Care :-)

CS61C L27 Single-Cycle CPU Control (21)
Boolean Expressions for Controller

RegDst        = add + sub
ALUSrc        = ori + lw + sw
MemtoReg      = lw
RegWrite      = add + sub + ori + lw
MemWrite      = sw
nPcsel        = beq
Jump          = jump
ExtOp         = lw + sw

ALUctr[0] = sub + beq (assume ALUctr is 0 ADD, 01: SUB, 10: OR)
ALUctr[1] = or

where,

rtype = \neg op_5 \cdot \neg op_4 \cdot \neg op_3 \cdot \neg op_2 \cdot \neg op_1 \cdot \neg op_0,
ori   = \neg op_5 \cdot \neg op_4 \cdot op_3 \cdot op_2 \cdot \neg op_1 \cdot op_0
lw    = op_5 \cdot \neg op_4 \cdot \neg op_3 \cdot \neg op_2 \cdot op_1 \cdot op_0
sw    = op_5 \cdot \neg op_4 \cdot op_3 \cdot \neg op_2 \cdot op_1 \cdot op_0
beq   = \neg op_5 \cdot \neg op_4 \cdot \neg op_3 \cdot op_2 \cdot \neg op_1 \cdot \neg op_0
jump  = \neg op_5 \cdot \neg op_4 \cdot \neg op_3 \cdot \neg op_2 \cdot op_1 \cdot \neg op_0

add = rtype \cdot func_5 \cdot \neg func_4 \cdot \neg func_3 \cdot \neg func_2 \cdot \neg func_1 \cdot \neg func_0
sub = rtype \cdot func_5 \cdot \neg func_4 \cdot \neg func_3 \cdot \neg func_2 \cdot func_1 \cdot \neg func_0
Controller Implementation

“AND” logic
- opcode
- func
- add
- sub
- ori
- lw
- sw
- beq
- jump

“OR” logic
- RegDst
- ALUSrc
- MemtoReg
- RegWrite
- MemWrite
- nPCsel
- Jump
- ExtOp
- ALUctr[0]
- ALUctr[1]
Peer Instruction

A. MemToReg=‘x’ & ALUctr=‘sub’. SUB or BEQ?

B. ALUctr=‘add’. Which 1 signal is different for all 3 of: ADD, LW, & SW? RegDst or ExtOp?

C. “Don’t Care” signals are useful because we can simplify our PLA personality matrix. F / T?

ABC
0: SRF
1: SRT
2: SEF
3: SET
4: BRF
5: BRT
6: BEF
7: BET
5 steps to design a processor

1. Analyze instruction set → datapath requirements
2. Select set of datapath components & establish clock methodology
3. Assemble datapath meeting the requirements
4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
5. Assemble the control logic
   - Formulate Logic Equations
   - Design Circuits
Bonus slides

• These are extra slides that used to be included in lecture notes, but have been moved to this, the “bonus” area to serve as a supplement.

• The slides will appear in the order they would have in the normal presentation.
The Single Cycle Datapath during Jump

- New PC = \{ PC[31..28], target address, 00 \}
The Single Cycle Datapath during Jump

- **New PC = { PC[31..28], target address, 00 }**

- New PC calculation:
  - $\text{PC}[31..28]$
  - Target address
  - 00 (no carry)

- Diagram details:
  - **RegDst = x**
  - **RegWr = 0**
  - **ALUctr = x**
  - **ExtOp = x**
  - **MemtoReg = x**
  - **Jump = 1**

- Instruction Fetch Unit:
  - **Instruction<31:0>**
  - $Rt$, $Rs$, $Rd$, $Imm16$, $TA26$
  - $MemWr = 0$
  - $Zero$

- 32 32-bit Registers:
  - $Rw$, $Ra$, $Rb$

- ALU Source and Destination:
  - $ALUSrc$, $ALUDst$

- Control Signals:
  - $nPC\_sel=?$
  - $Jump=1$
  - $MemtoReg = x$

- Memory Access:
  - $WrEn$, $Adr$, $Data$

- Cycles and States:
  - New PC calculation cycle
  - Instruction fetch cycle
  - ALU computation cycle
Instruction Fetch Unit at the End of Jump

- **New PC = \{ PC[31..28], target address, 00 \}**

How do we modify this to account for jumps?
Instruction Fetch Unit at the End of Jump

- New PC = \{ PC[31..28], target address, 00 \}

Query
- Can Zero still get asserted?
- Does nPC_sel need to be 0?
  - If not, what?