

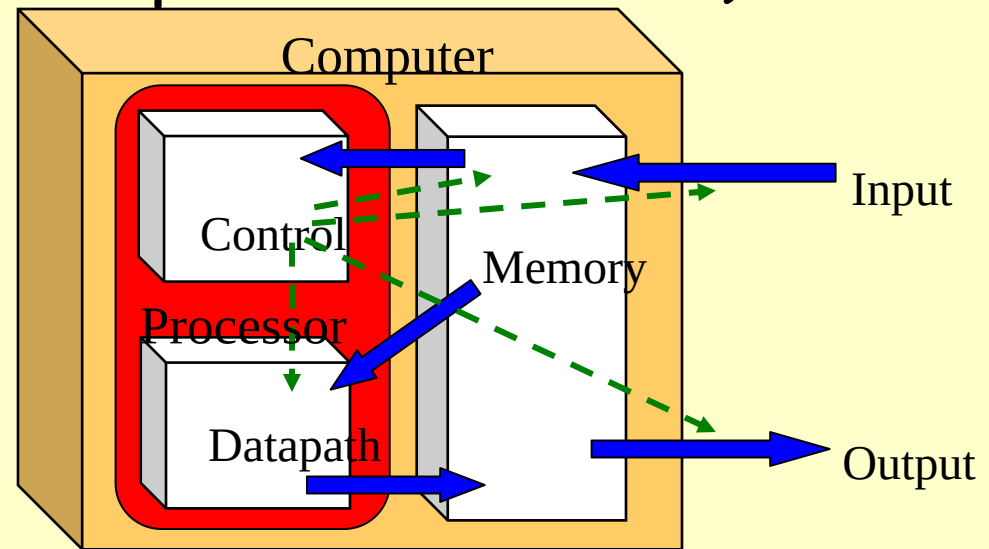
Chapter 5: "The Processor: Datapath and Control"

Part One, The Single Cycle Processor

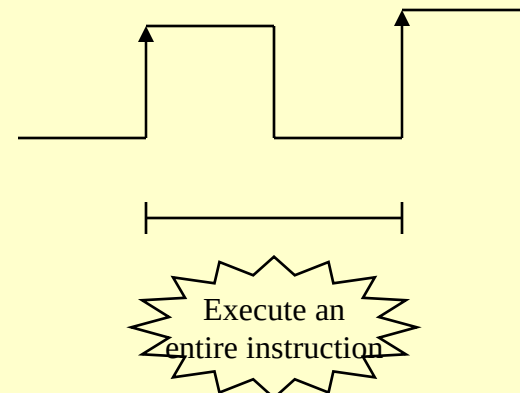
The Performance Big Picture

- Execution Time = Insts * CPI * Cycle Time
- Processor design (datapath and control) will determine:

- Clock cycle time
- Clock cycles per instruction



- Starting today:
 - Single cycle processor:
 - Advantage: CPI = 1
 - Disadvantage: long cycle time

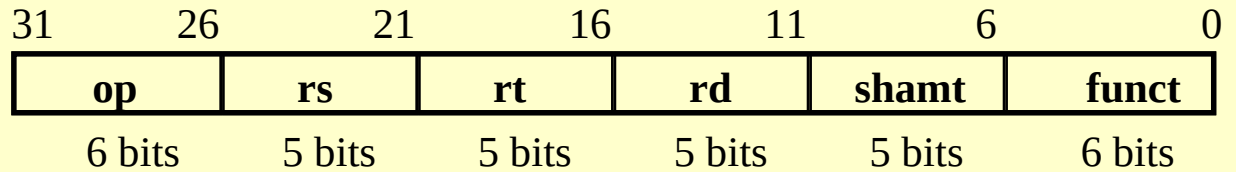


What parts of MIPS?

- We'll cover a subset of MIPS
 - Memory instructions
 - Arithmetic/Logical (and just a subset of these, but you should be able to figure out how to add many of them)
 - BEQ and J
 - Basic load/store architecture with these steps:
 - Read PC and Fetch Inst
 - Read Registers
 - Do Operation
 - Write memory/registers
 - Repeat
- .. but you should be able to extend what we do to handle more of the instructions

The MIPS core subset

• R-type

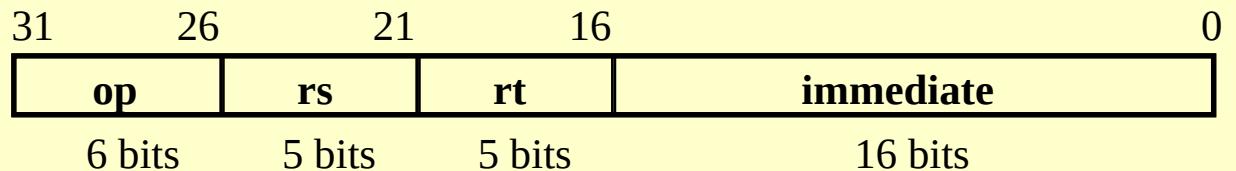


- *add rd, rs, rt*

- *sub, and, or, slt*

1. Read registers rs and rt
2. Feed them to ALU
3. Update register file

• LD/ST

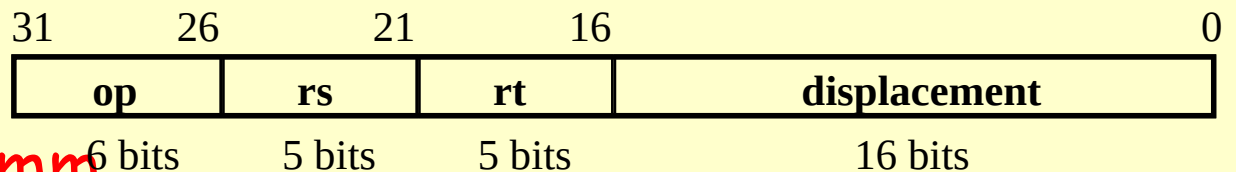


- *lw rt, rs, imm*

- *sw rt, rs, imm*

1. Read register rs (and rt for store)
2. Feed rs and immed to ALU
3. Move data between mem and reg

• BRANCH:



- *beq rs, rt, imm*

1. Read registers rs and rt
2. Feed to ALU to compare
3. Add PC to disp; update PC

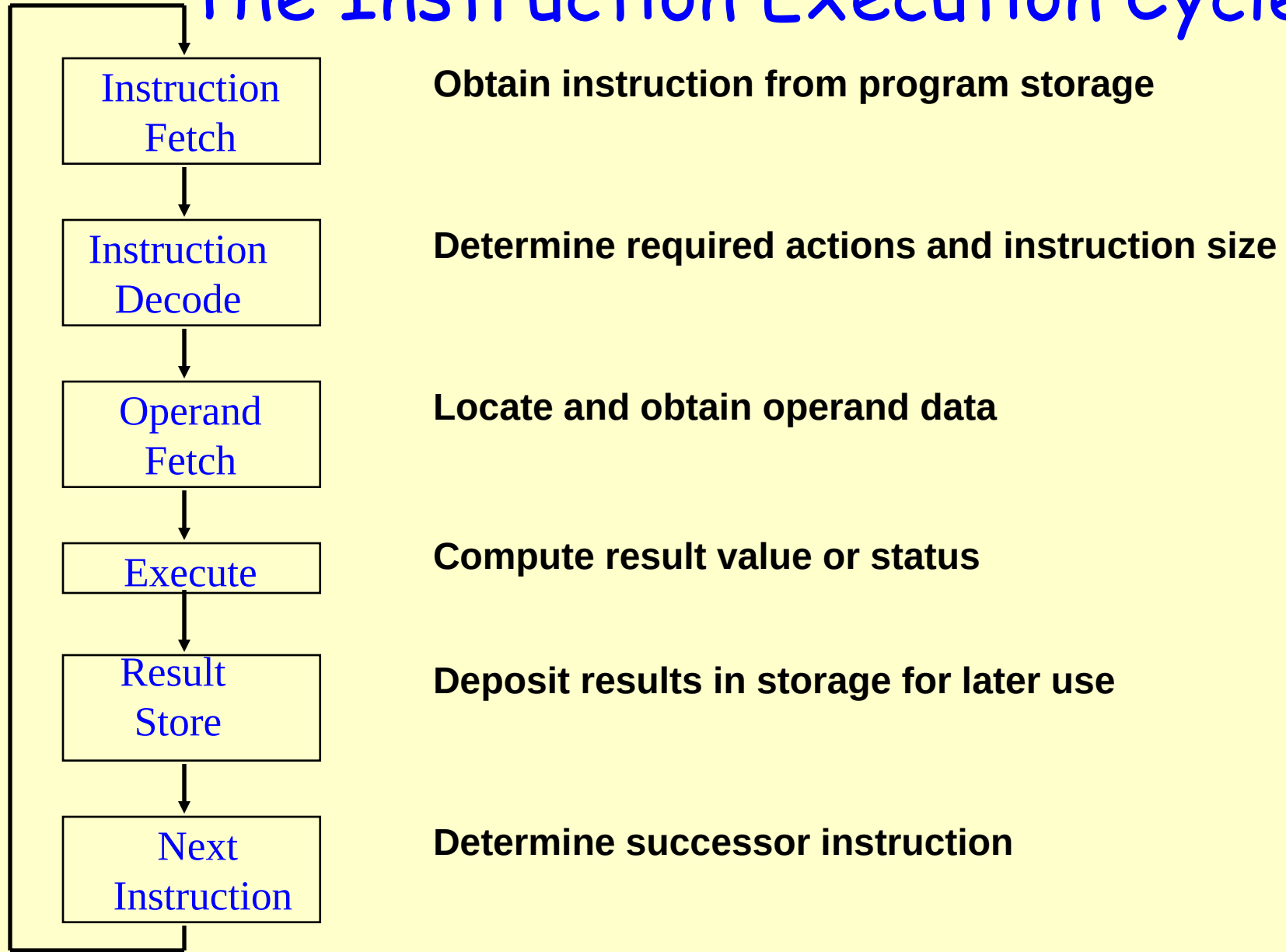
Register Transfer Language (RTL)

- Is a mechanism for describing the movement of data between storage elements
 - Gives us a precise way to describe various actions of our instructions
 - May be more than 1 RTL statement per inst
 - $R[3] \leftarrow R[5] + R[7]$
 - $PC \leftarrow PC + 4$
 - $R[rd] \leftarrow R[rs] + R[rt]$

The basic design algorithm (after you have the ISA) - you'll do this for your own ISA for 141L

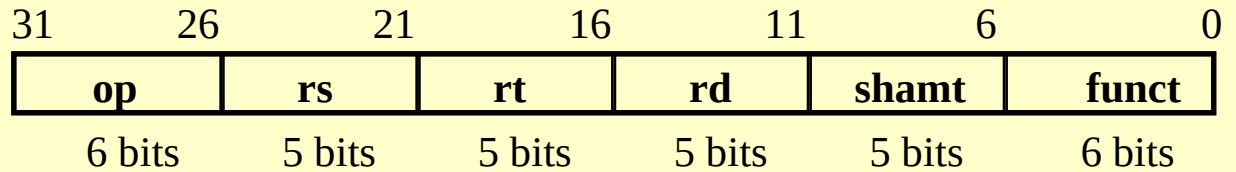
- Build the datapath on the whiteboard
 - one by one,
simulate each instruction
on the current datapath "sketch":
 - make sure it is workable
 - if not, modify datapath
- Design the control logic
 - one by one,
simulate each instruction on the current
datapath + control logic:
 - make sure it is workable
 - if not, modify control or datapath

The Instruction Execution Cycle



The MIPS core subset

• R-type

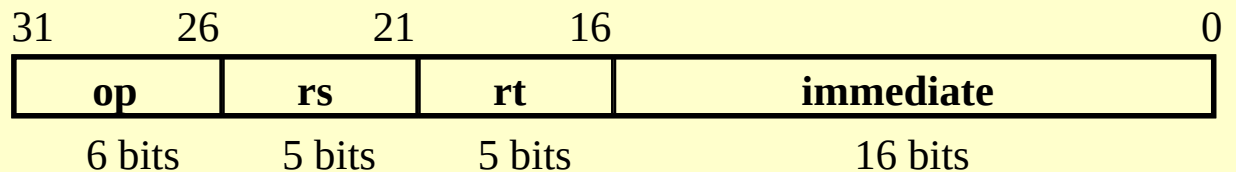


- *add rd, rs, rt*

- *sub, and, or, slt*

1. Read registers rs and rt
2. Feed them to ALU
3. Update register file

• LD/ST

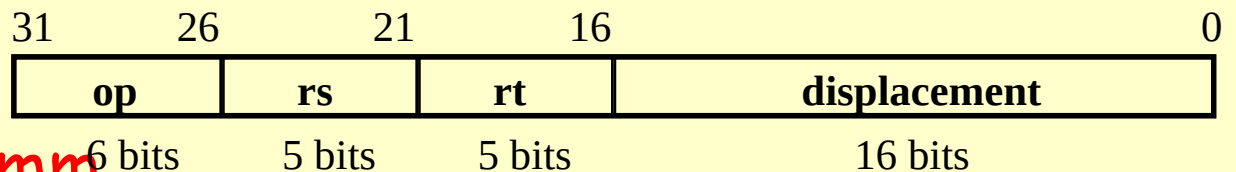


- *lw rt, rs, imm*

- *sw rt, rs, imm*

1. Read register rs (and rt for store)
2. Feed rs and immed to ALU
3. Move data between mem and reg

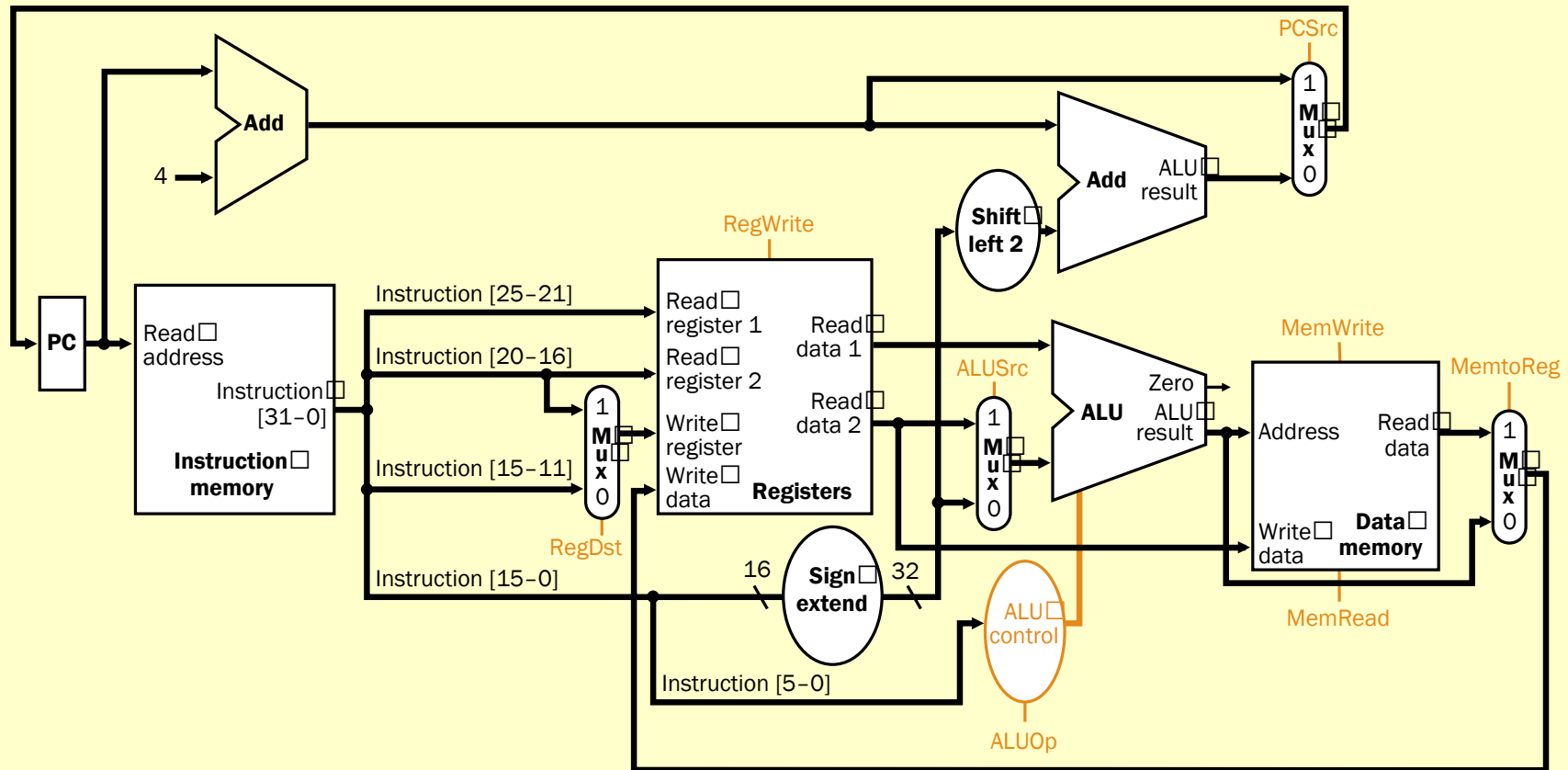
• BRANCH:



- *beq rs, rt, imm*

1. Read registers rs and rt
2. Feed to ALU to compare
3. Add PC to disp; update PC

R-Format/ Lw/ Sw/ BEQ



	ALUsrc	ALUop	MemRead	MemWrite	MemToReg	RegDst	RegWrite	PCsrc
RRR	1	tbd	0	0	0	0	1	:
lw	0	tbd	1	0	1	1	1 ₉	:
bne	1	cmp	0	0	X	X	0	"Zero"

```

opc    = Inst[31:26];    func    = Inst[31:26];
rs     = Inst[25:21];    rt      = Inst[20:16];
rd     = Inst[15:11];    sa      = Inst[10:6];
imm    = Inst[15:0];

```

```

WritesRD = (opc == 'OP_RRR)
WritesRT = ( (opc == 'OP_ADDI) || (opc == 'OP_ADDIU)
             || (opc == 'OP_ANDI) || (opc == 'OP_LBU)
             || (opc == 'OP_LHU)  || (opc == 'OP_LW)
             || (opc == 'OP_LUI)  || (opc == 'OP_ORI)
             || (opc == 'OP_SLTI) || (opc == 'OP_SLTIU));

IsStore = ( (opc == 'OP_SB)  || (opc == 'OP_SH)
            || (opc == 'OP_SW));

IsLoad  = ( (opc == 'OP_LBU) || (opc == 'OP_LHU)
            || (opc == 'OP_LW));

ReadsRT = ( (opc == 'OP_RRR) || (opc == 'OP_BEQ)
            || (opc == 'OP_BNE) || (IsStore);

```

```
RegDst    = WritesRD;  
RegWrite  = (WritesRD | WritesRT);  
MemWrite  = IsStore;  
MemRead   = IsLoad;  
MemToReg  = IsLoad;
```

```
ALUSrc    = ReadsRT & ~IsStore;  
           // careful - some diagrams seem  
           // inconsistent about ALUSrc "mux sense"
```

```
PCSrc     = IsBranch && BranchTaken; //from ALU
```

```
IsBranch  = ( (opc == 'OP_BEQ) || (opc == 'OP_BNE)  
             || (opc == 'OP_BLEZ) || (opc == 'OP_BGTZ)  
             || ..  
             )
```

```
ALUOp    =           // conversion between opc/func and muxes  
           // inside ALU
```