# Processor Design

## ELEC 418
## Advanced Digital Systems
## Dr. Ron Hayne

*Images Courtesy of Thomson Engineering*

# 68HC11 Programming Model

♦ **Motorola 68HC11 Microcomputer**

| 7 | A | 0 | 7 | B | 0 | 8-bit Accumulators A & B |
|---|---|---|---|---|---|---|
| 15 | | | D | | 0 | 16-bit Double Accumulator D |

| 15 | X | 0 | Index Register X |
|----|---|---|---|

| 15 | Y | 0 | Index Register Y |
|----|---|---|---|

| 15 | SP | 0 | Stack Pointer |
|----|----|---|---|

| 15 | PC | 0 | Program Counter |
|----|----|---|---|

| S | X | H | I | N | Z | V | C | Condition Code Register |
|---|---|---|---|---|---|---|---|---|

# 68HC11 Instruction Set Table

| Source Form | Operation | Boolean Expression | Addr. Mode | Machine Code | | Bytes | Cycles |
|---|---|---|---|---|---|---|---|
| | | | | Op Code | Op-er and | | |
| ABX | Add B to X | $X + 00{:}B \rightarrow X$ | INH | 3A | | 1 | 3 |
| • | • | • | • | • | • | • | • |
| ADDA (opr) | Add Memory to A | $A + M \rightarrow A$ | A IMM<br>A DIR<br>A EXT<br>A IND,X<br>A IND,Y | 8B<br>9B<br>BB<br>AB<br>18 AB | ii<br>dd<br>hh ll<br>ff<br>ff | 2<br>2<br>3<br>2<br>3 | 2<br>3<br>4<br>4<br>5 |
| • | • | • | • | • | • | • | • |
| CLC | Clear Carry Bit | $0 \rightarrow C$ | INH | 0C | | 1 | 2 |
| • | • | • | • | • | • | • | • |
| LDX (opr) | Load Index Register X | $M{:}(M + 1) \rightarrow X$ | X IMM<br>X DIR | CE<br>DE | jj kk<br>dd | 3<br>2 | 3<br>4 |

# PIC18F – MPU and Memory
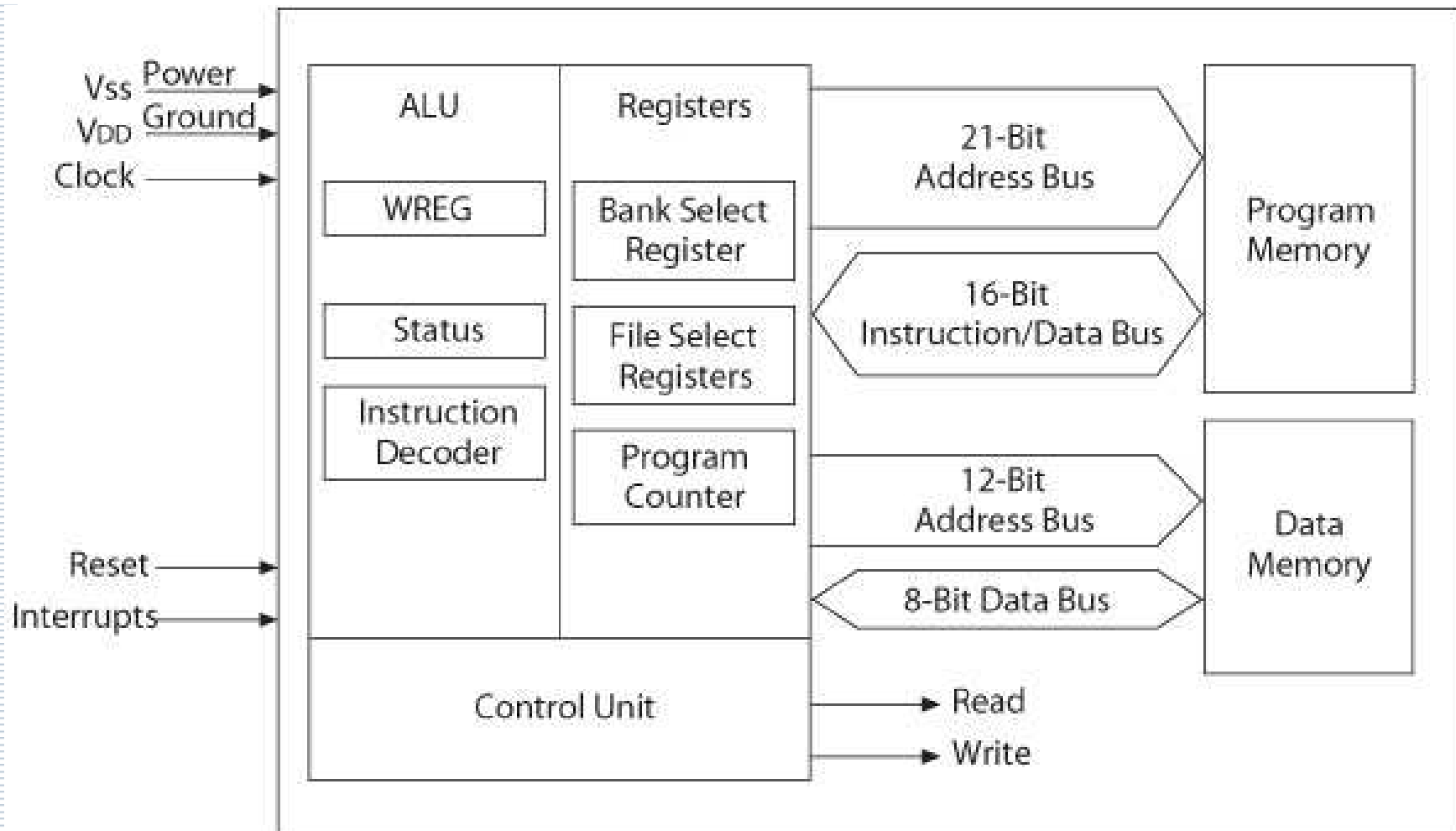
# PIC18F4X2 Architecture Block Diagram
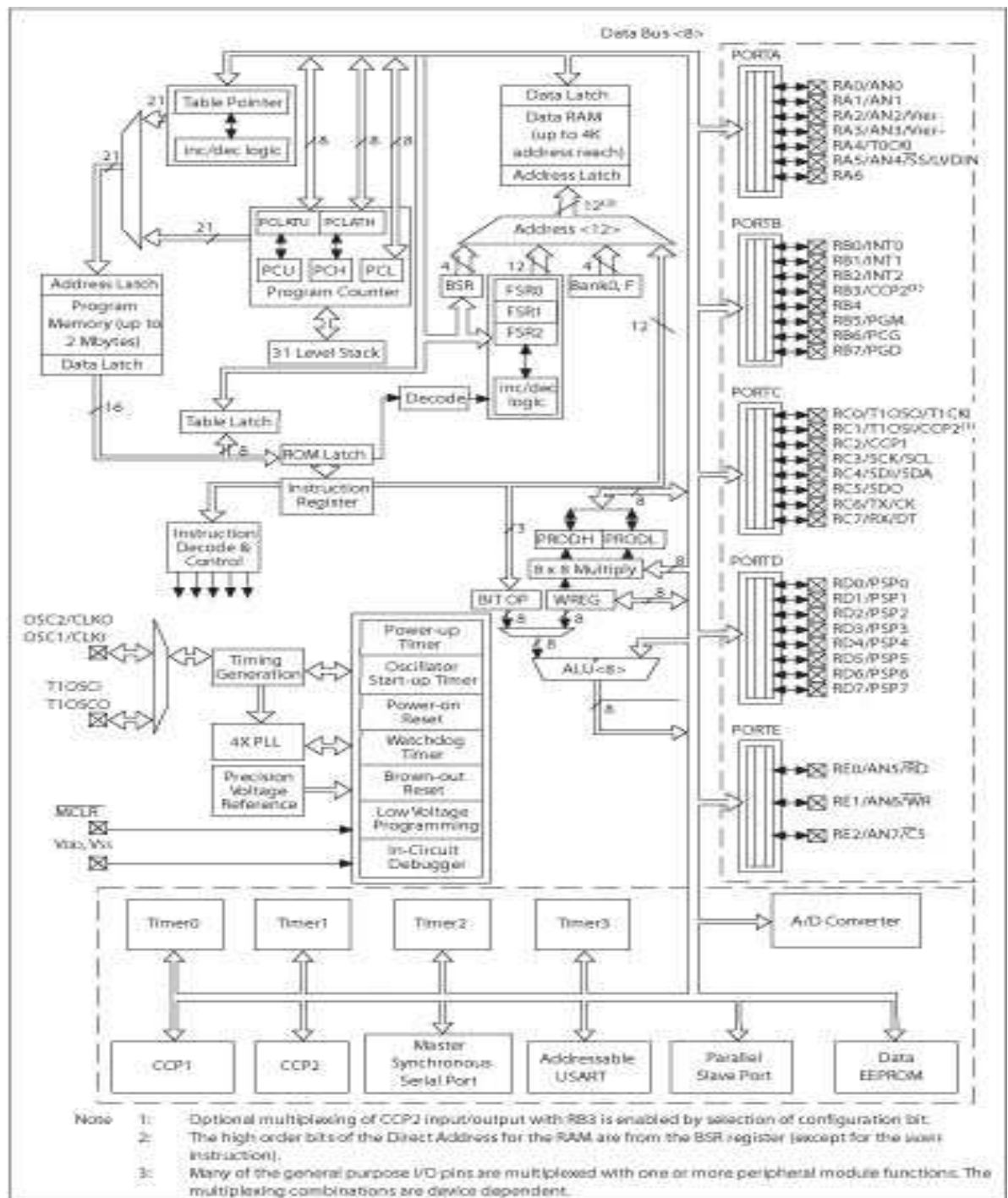
# MIPS ISA

- **Instruction Set Architecture**
  - 32 General-Purpose Registers (32-bits)
  - 3 Instruction Formats
    - R-format (register)
    - I-format (immediate)
    - J-format (jump)
  - 1 Addressing Mode
    - Base register and signed offset

FIGURE 9-3. **Overall Data Path**
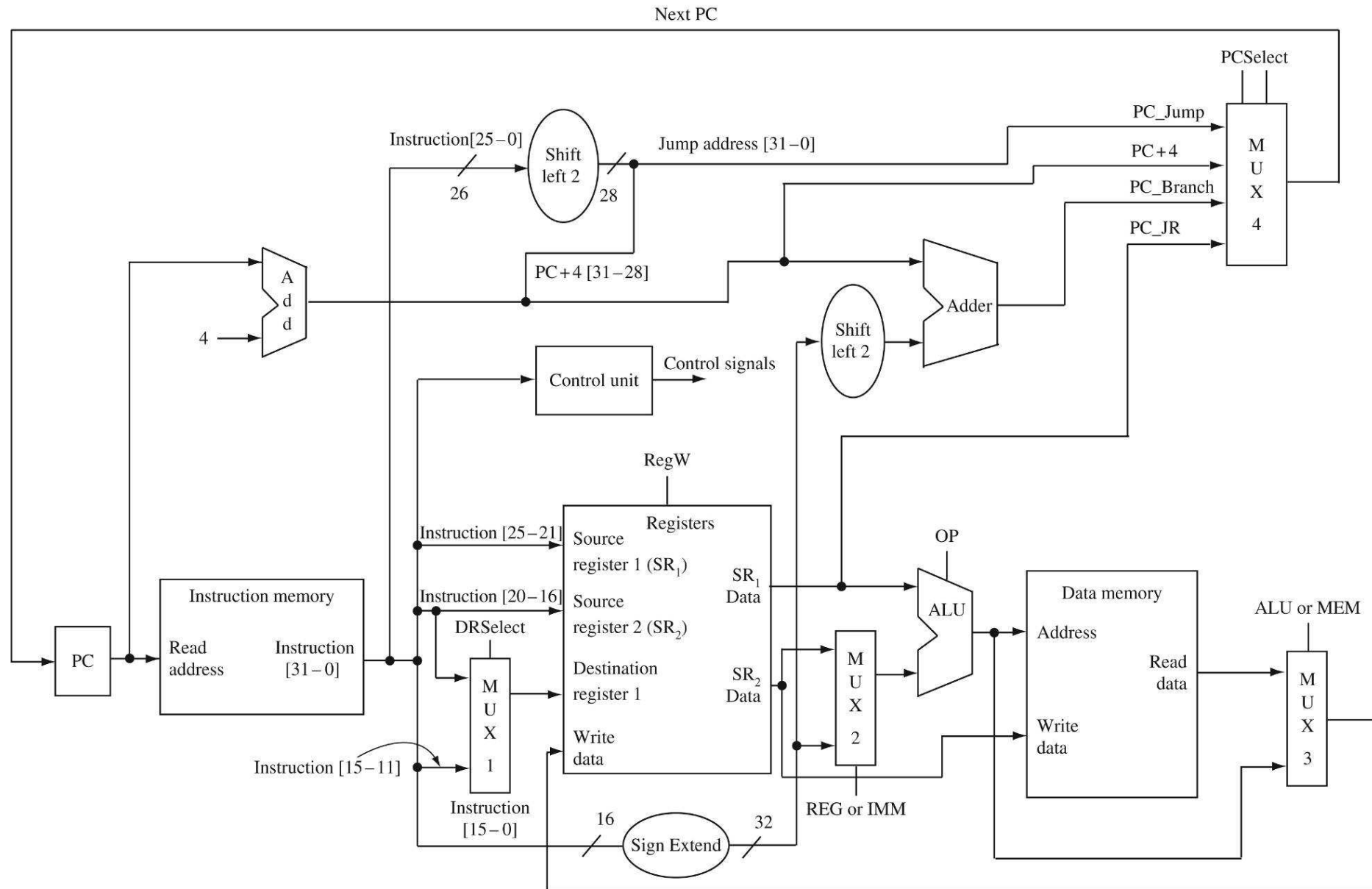
# Data Path Design

## MIPS Subset

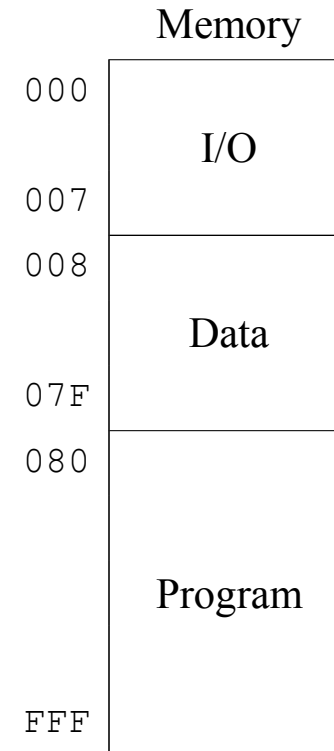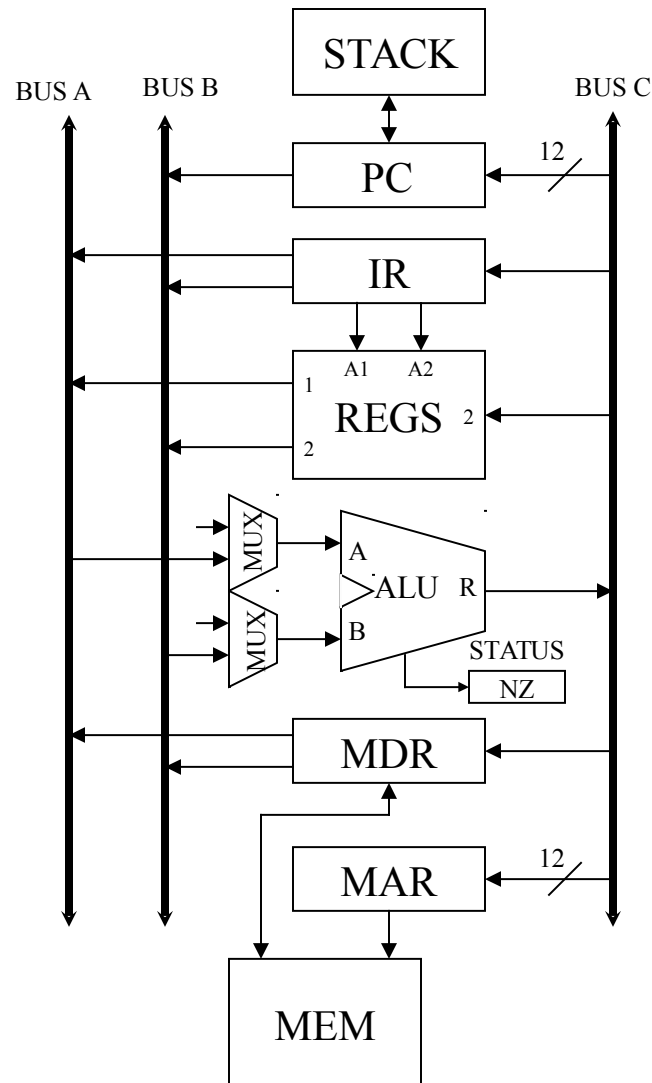| | |
|---|---|
| Arithmetic | add<br>subtract<br>add immediate |
| Logical | and<br>or<br>and immediate<br>or immediate<br>shift left logical<br>shift right logical |
| Data Transfer | load word<br>store word |
| Conditional branch | branch on equal<br>branch on not equal<br>set on less than |
| Unconditional branch | jump<br>jump register |

## Sequence of Operations

- Fetch an Instruction
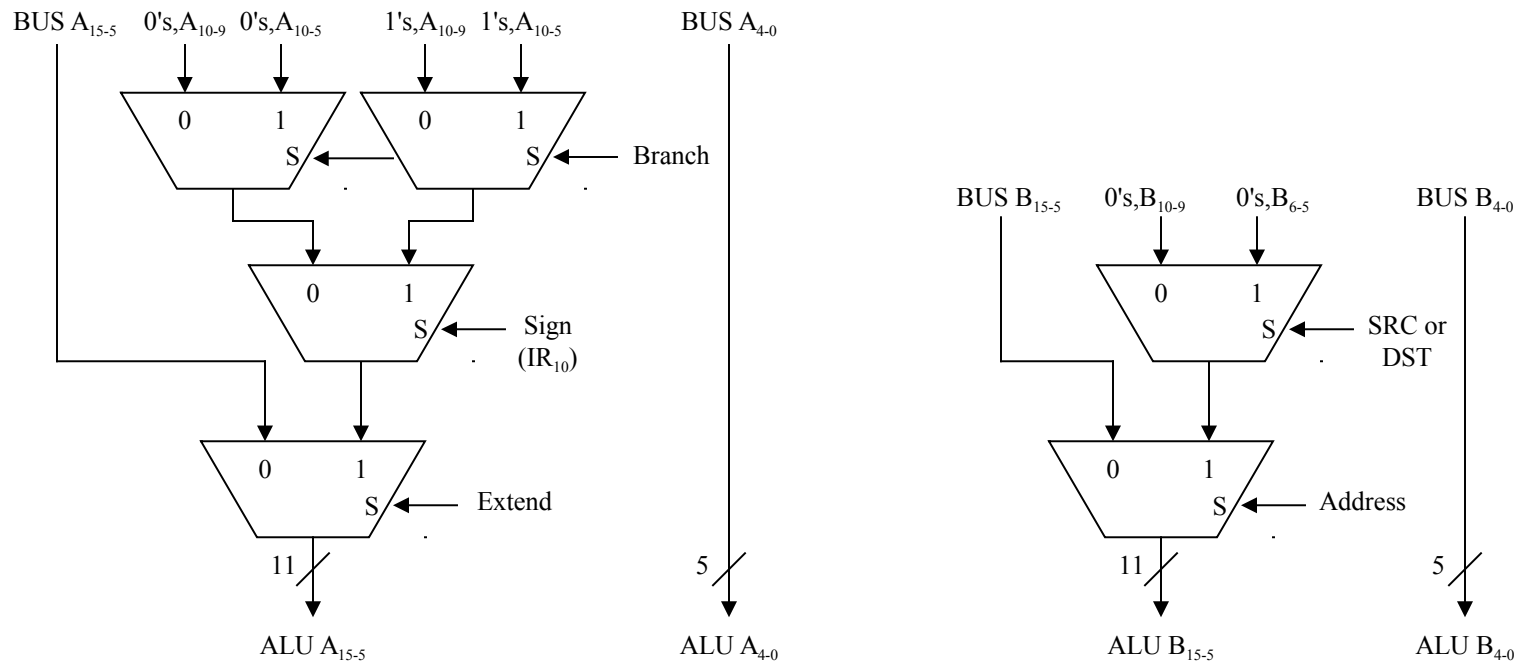- Decode the Instruction
- Execute the Instruction

# Instructional Processor Design

- **3 Bus Organization**
  - 16 bit Data Path
- **4 Word Register File**
- **4K Word Memory**
- **8 Function ALU**
  - 2 Condition Code Flags
- **5 Data Instructions**
  - 4 Addressing Modes
- **4 Branch Instructions**

# Data Path & Memory

# ALU Multiplexers



BUS A$_{15-5}$  0's,A$_{10-9}$  0's,A$_{10-5}$  1's,A$_{10-9}$  1's,A$_{10-5}$  BUS A$_{4-0}$

Branch

Sign (IR$_{10}$)

Extend

11

5

ALU A$_{15-5}$

ALU A$_{4-0}$

BUS B$_{15-5}$  0's,B$_{10-9}$  0's,B$_{6-5}$  BUS B$_{4-0}$

SRC or DST

Address

11

5

ALU B$_{15-5}$

ALU B$_{4-0}$

# Data Path Registers & Memory

- **Program Counter (PC)**
  - 12-bit Program Address
- **Subroutine Stack (STACK)**
  - 16 x 12-bit Addresses
- **Instruction Register (IR)**
  - 16-bit Instructions
- **Register File (REGS)**
  - 4 x 16-bit Registers
- **Arithmetic Logic Unit (ALU)**
  - 8 Functions (ALU_OP)
- **Flag Register (STATUS)**
  - Negative Flag (N)
  - Zero Flag (Z)

- **Memory Data Register (MDR)**
  - 16-bits to/from Memory
- **Memory Address Reg (MAR)**
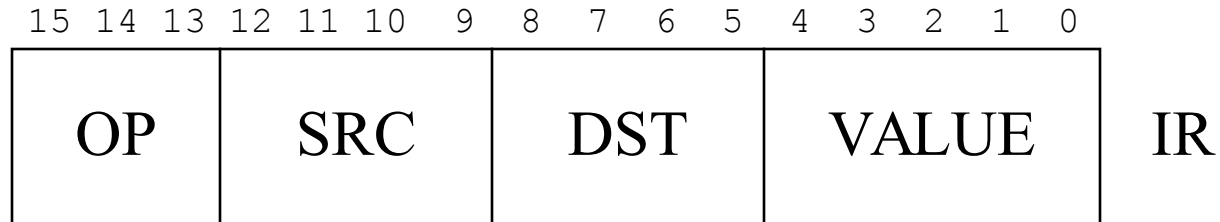  - 12-bit Memory Address
- **Memory (MEM)**
  - 4K x 16-bit Memory

# Memory Map

♦ **`4K (4096) RAM`**

 ▪ 8 Memory-mapped I/O Ports

  • 0x000      Switch      (Input)

  • 0x001      LED      (Output)

 ▪ 120 Data Memory Locations

  • 0x008 - 0x07F

 ▪ 3968 Program Memory Locations

  • 0x080 - 0xFFF

# Addressing Modes

- **Method of specifying of an operand**
  - Immediate (Literal) addressing
    - The operand is a number that follows the opcode
  - Direct (Absolute) addressing
    - The address of the operand is a part of the instruction
  - Indirect addressing
    - An address is specified in a register (pointer) and the MPU looks up the address in that register

# Data Instruction Format

```
15 14 13 12 11 10  9   8   7   6   5   4   3   2   1   0
```

| OP | SRC | DST | VALUE | IR |
|----|-----|-----|-------|----|

| | Mode | REG # | Name | Syntax | Effective Address |
|---|------|-------|------|--------|-------------------|
| SRC or DST | 00 | 00-11 | Register Direct | Rn | EA = Rn |
| | 01 | 00-11 | Register Indirect | (Rn) | EA = [Rn] |
| | 10 | vv | Absolute | Value | EA = Value |
| | 11* | vv | Immediate | #Value | Operand = Value |

EA = Effective Address
vv  = Upper 2 bits of Value
* = SRC only

# Data Instructions

| OP | Fn | Assembly Language | Register Transfer Notation (RTN) |
|-----|------|-------------------|----------------------------------|
| 000 | MOVE | MOVE SRC,DST | DST ← SRC |
| 001 | ADD | ADD  SRC,DST | DST ← SRC + DST |
| 010 | INV | INV  SRC,DST | DST ← not SRC |
| 011 | AND | AND  SRC,DST | DST ← SRC and DST |
| 100 | ROTL | ROTL SRC,DST | DST ← SRC(14 dt 0) & SRC(15) |
| ... | | | |

# Branch Instruction Format

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
```

| OP | MD | OFFSET | IR |
|----|----|--------|----|

| OP | MD | Fn | Assy Lang | RTN |
|----|----|-----|-----------|-----|
| 111 | 00 | BRA | `BRA  Offset` | PC ← PC + Offset |
| | 01 | BGTZ | `BGTZ Offset` | PC ← PC + Offset  (STATUS > 0) |
| | 10 | BSR | `BSR  Offset` | STACK ← PC; PC ← PC + Offset |
| | 11 | RTN | `RTN` | PC ← STACK |

# Assembly Language Program

```
        MOVE N,R1
        MOVE #NUM1,R2
        MOVE #0,R0
LOOP ADD    (R2),R0
        ADD   #1,R2
        ADD   #-1,R1
        BGTZ LOOP
        MOVE R0,SUM
STOP BRA   STOP
```

| | |
|---|---|
| | . . . |
| SUM | |
| N | n |
| NUM1 | |
| NUM2 | |
| | . . . |
| NUMn | |
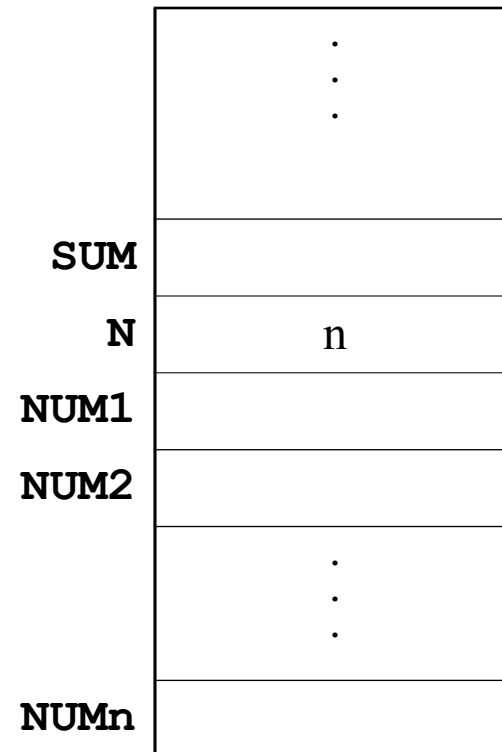
# Control Unit Organization

CLK → Control Step Counter

Step DCD

$T_0$ ... $T_7$

IR

Instruction DCD — $I_0$ ... $I_7$

SRC_MODE DCD — $M_0$ ... $M_3$

DST_MODE DCD — $M_0$ ... $M_3$

Encoder

N ... C ← STATUS

Clear
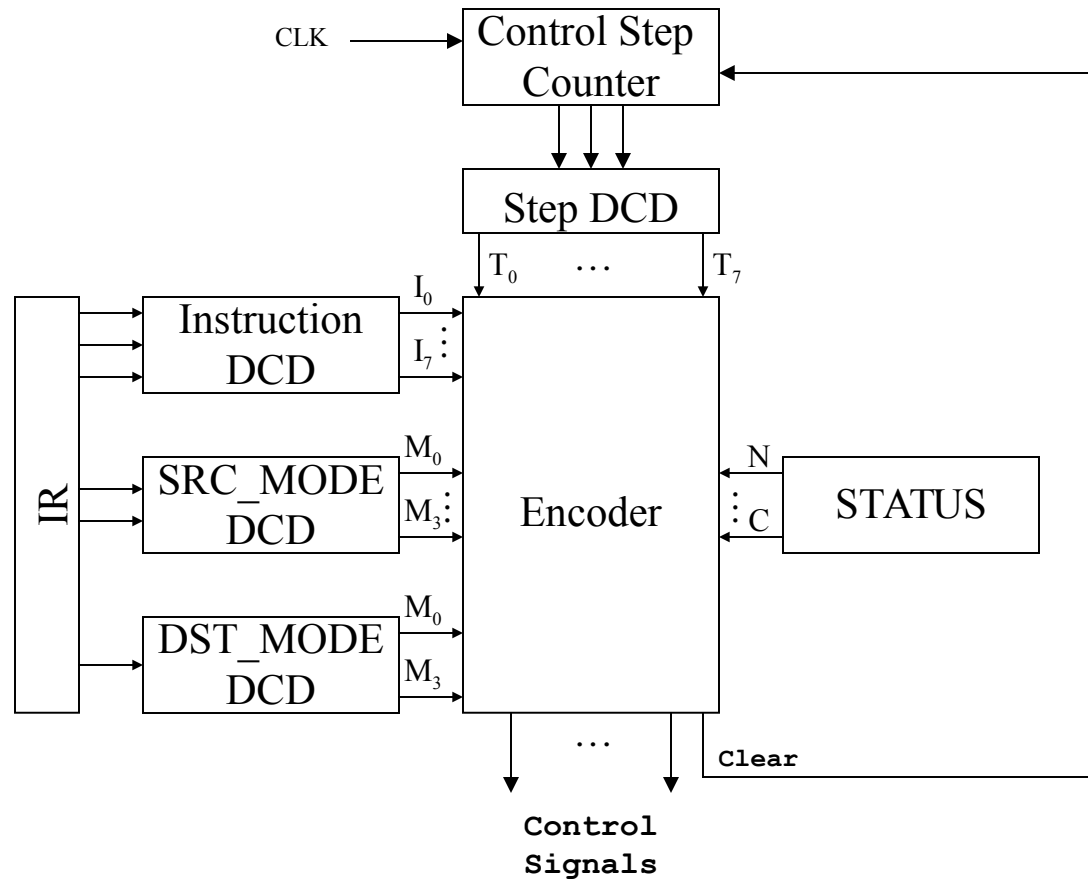
**Control Signals**

# Control Signals

- **BUS_A**
- **BUS_B**
- **REGS_Read1**
- **REGS_Read2**
- **Extend**
- **Address**
- **ALU_Op**
- **MEM_Read**
- **MEM_Write**

- **Inc_PC**
- **Load_PC**
- **Push_PC**
- **Pop_PC**
- **Load_IR**
- **REGS_Write**
- **Load_STATUS**
- **Load_MDR**
- **Load_MAR**
- **Clear**

# Control Unit Design

♦ **Instruction Fetch**

| Step | RTN | Control Signals |
|------|-----|-----------------|
| T0 | MAR ← PC, PC ← PC + 1 | BUS_B <= PC<br>ALU_OP <= Pass_B<br>Load_MAR <= '1'<br>Inc_PC <= '1' |
| T1 | MDR ← MEM(MAR) | MEM_Read <= '1'<br>Load_MDR <= '1' |
| T2 | IR ← MDR | BUS_B <= MDR<br>ALU_OP <= Pass_B<br>Load_IR <= '1' |

# Control Unit Design

♦ **Instruction Execute**

■ MOVE Rs,Rd

• Register Direct (M0), Register Direct (M0)

| Step | RTN | Control Signals |
|------|-----|-----------------|
| T3 | R(D) ← R(S) | REGS_Read1 <= '1'<br>ALU_OP <= Pass_A<br>Load_STATUS <= '1'<br>REGS_Write <= '1'<br>Clear <= '1' |

# Control Unit Design

♦ **Instruction Execute**

- ▪ `MOVE Value,Rd`
    - Immediate (M3), Register Direct (M0)

| Step | RTN | Control Signals |
|------|-----|-----------------|
| T3 | R(D) ← Value | BUS_A <= IR<br>Extend <= '1'<br>ALU_OP <= Pass_A<br>Load_STATUS <= '1'<br>REGS_Write <= '1'<br>Clear <= '1' |

# Control Unit Design

- **Instruction Execute**
  - BRA Offset
    - Branch Always
    - OP (111), Mode (M0)

| Step | RTN | Control Signals |
|------|-----|-----------------|
| T3 | PC ← PC + Offset | BUS_A <= IR<br>BUS_B <= PC<br>Extend <= '1'<br>ALU_OP <= ADD<br>Load_PC <= '1'<br>Clear <= '1' |

# VHDL Model (Phase 1)

- **Data Path**
  - Components
- **Control Unit**
  - Instruction Fetch
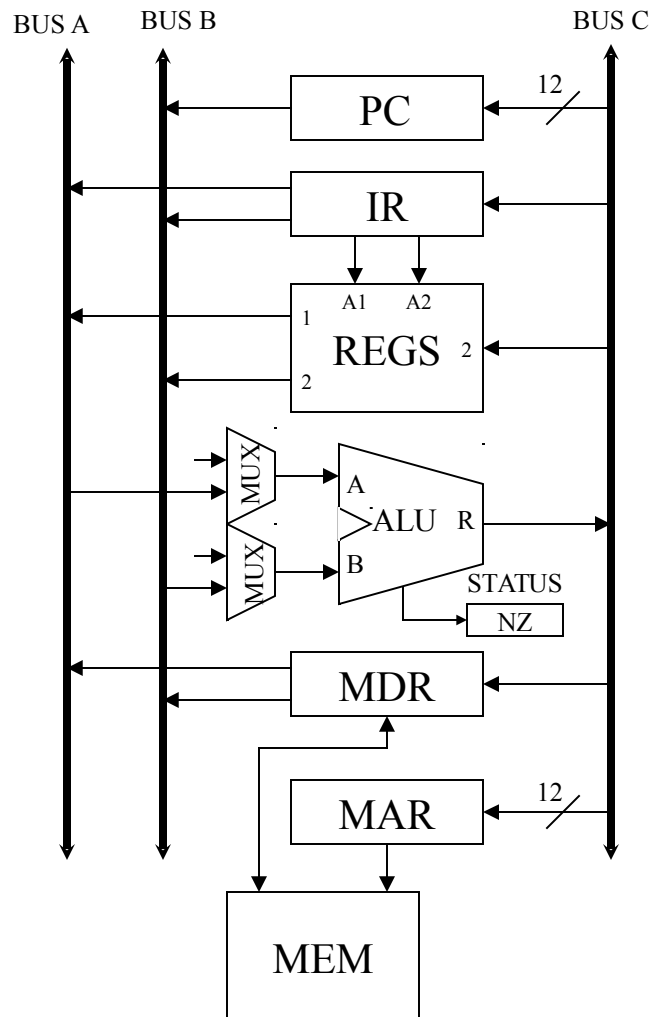  - Instruction Execute

- **First Test Program**
  - program1.asm
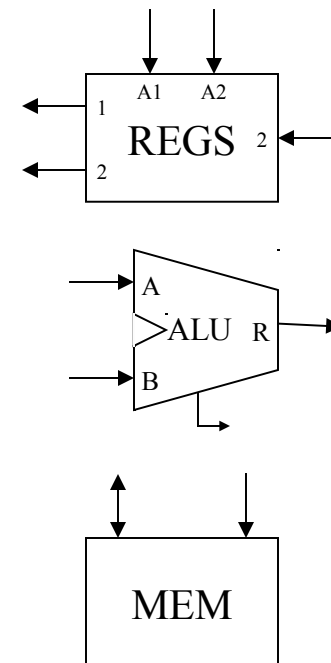  - program1.bin

```
     MOVE #3,R1
     MOVE R1,R2
STOP BRA  STOP
```

# VHDL Model (Phase 1)



- **processor1.vhd**
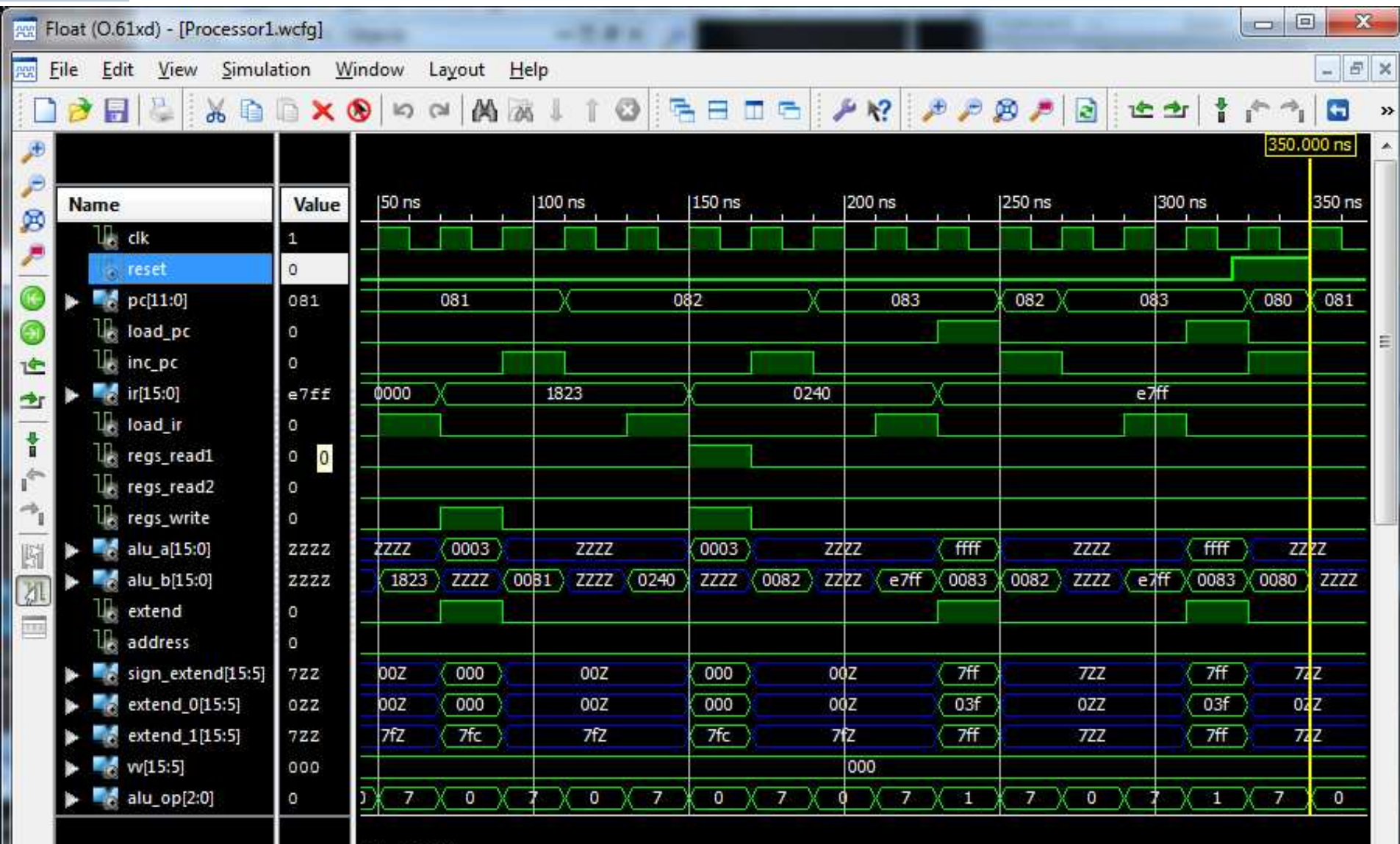- **processor1_component s.vhd**

# VHDL Testbench

```vhdl
constant CLK_period : time := 20 ns;

stim_proc : process
  begin
    RESET <= '1';
    wait for CLK_period*1.25;
    RESET <= '0';
    wait for CLK_period*15;
end process;
```

# VHDL Simulation (Phase 1)

# Control Unit Design (Phase 2)

- MOVE Rs,Addr
  - Register Direct (M0), Absolute (M2)

| Step | RTN | Control Signals |
|------|-----|-----------------|
| T3 | MDR ← R(S) | REGS_Read1 <= '1'<br>ALU_OP <= Pass_A<br>Load_STATUS <= '1'<br>Load_MDR <= '1' |
| T4 | MAR ← Value | BUS_B <= IR;<br>Address <= '1'<br>ALU_OP <= Pass_B<br>Load_MAR <= '1' |
| T5 | MEM(MAR) ← MDR | MEM_Write <= '1'<br>Clear <= '1' |

# Control Unit Design (Phase 2)

- `MOVE Addr,Rd`
  - Absolute (M2), Register Direct (M0)

| Step | RTN | Control Signals |
|------|-----|-----------------|
| T3 | MAR ← Value | BUS_B <= IR<br>Address <= '1'<br>ALU_OP <= Pass_B<br>Load_MAR <= '1' |
| T4 | MDR ← MEM(MAR) | MEM_Read <= '1'<br>Load_MDR <= '1' |
| T5 | R(D) ← MDR | BUS_B <= MDR<br>ALU_OP <= Pass_B<br>Load_STATUS <= '1'<br>REGS_Write <= '1'<br>Clear <= '1' |

# Control Unit Design (Phase 2)

- ADD (Rs),Rd
  - Register Indirect (M1), Register Direct (M0)

| Step | RTN | Control Signals |
|------|-----|-----------------|
| T3 | MAR ← R(S) | REGS_Read1 <= '1'<br>ALU_OP <= Pass_A<br>Load_MAR <= '1' |
| T4 | MDR ← MEM(MAR) | MEM_Read <= '1'<br>Load_MDR <= '1' |
| T5 | R(D) ← MDR + R(D) | BUS_A <= MDR<br>REGS_Read2 <= '1'<br>ALU_OP <= OP<br>Load_STATUS <= '1'<br>REGS_Write <= '1'<br>Clear <= '1' |

# Control Unit Design (Phase 2)

- ADD #Value,Rd
  - Immediate (M3), Register Direct (M0)

| Step | RTN | Control Signals |
|------|-----|-----------------|
| T3 | $R(D) \leftarrow Value + R(D)$ | BUS_A <= IR<br>Extend <= '1'<br>REGS_Read2 <= '1'<br>ALU_OP <= OP<br>Load_STATUS <= '1'<br>REGS_Write <= '1'<br>Clear <= '1' |

# Control Unit Design (Phase 2)

- BGTZ Offset
  - Branch if greater than zero
  - OP (111), Mode (M1)

| Step | RTN | Control Signals |
|------|-----|-----------------|
| T3 | if N = 0 and Z = 0 then<br>    PC ← PC + Offset | BUS_A <= IR<br>BUS_B <= PC<br>Extend <= '1'<br>ALU_OP <= ADD<br>Load_PC <= '1'<br>Clear <= '1' |

# Assembly Language Program

- **program.asm**
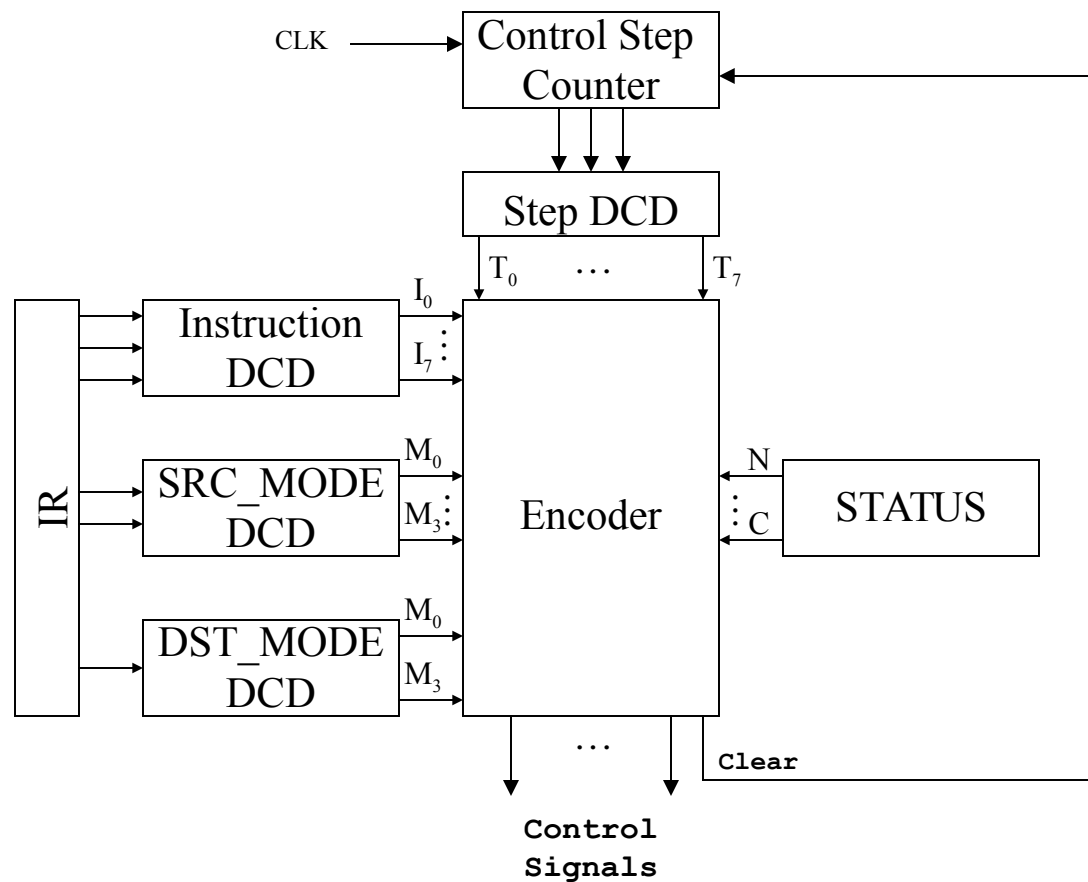- **program.bin**

```
        MOVE N,R1
        MOVE #NUM1,R2
        MOVE #0,R0
  LOOP  ADD   (R2),R0
        ADD   #1,R2
        ADD   #-1,R1
        BGTZ  LOOP
        MOVE  R0,SUM
  STOP  BRA   STOP
```
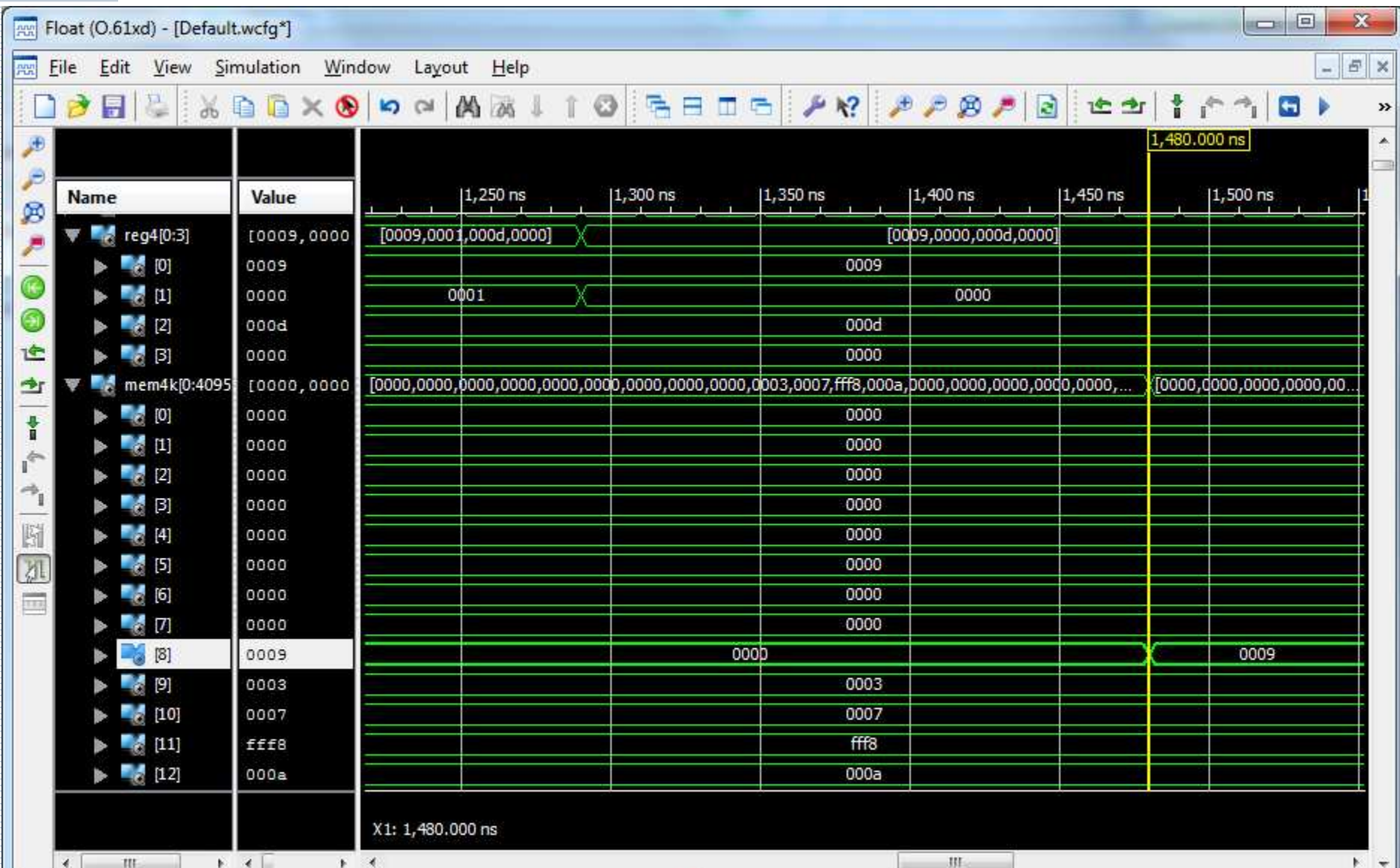
| | |
|---|---|
| | . . . |
| SUM | |
| N | n |
| NUM1 | |
| NUM2 | |
| | . . . |
| NUMn | |
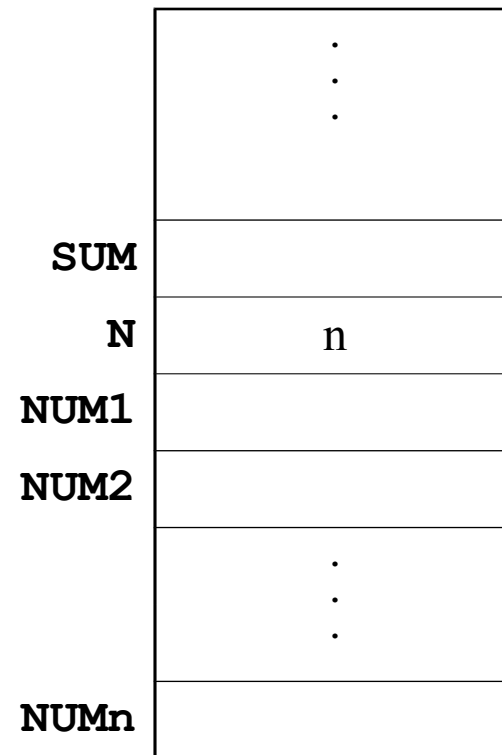
# VHDL Control Unit (Phase 2)

- **processor2.vhd**

# VHDL Simulation (Phase 2)

# Microcontroller (Phase 3)

- **`4K (4096) RAM`**
  - 8 Memory-mapped I/O Ports
    - 0x000      SWITCH      (Input)      8-bit
    - 0x001      LED      (Output)      8-bit
    - 0x002      ANODE      (Output)      4-bit
    - 0x003      CATHODE      (Output)      8-bit
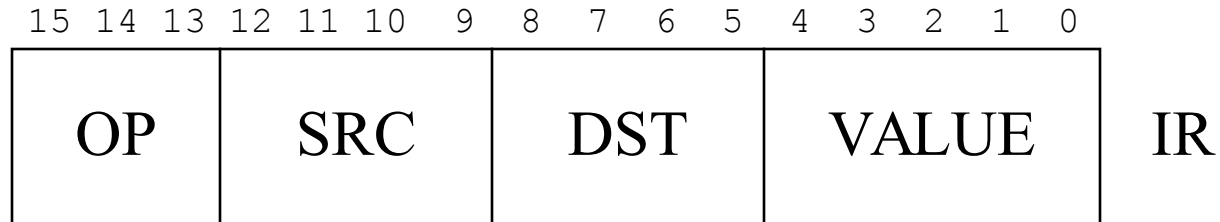  - 120 Data Memory Locations
    - 0x008 - 0x07F

# MEM4K

```vhdl
entity MEM4K is
  port(CLK: in std_logic;
        MEM_Read: in std_logic;
        MEM_Write: in std_logic;
        Addr: in std_logic_vector(11 downto 0);
        Data_In: in std_logic_vector(15 downto 0);
        Data_Out: out std_logic_vector(15 downto 0);
        SWITCH: in std_logic_vector(7 downto 0);
        LED: out std_logic_vector(7 downto 0);
        ANODE: out std_logic_vector(3 downto 0);
        CATHODE: out std_logic_vector(7 downto 0));
end MEM4K;
```

# Data Instruction Format

```
15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
```

| OP | SRC | DST | VALUE | IR |
|----|-----|-----|-------|-----|

| | Mode | REG # | Name | Syntax | Effective Address |
|---|------|-------|------|--------|-------------------|
| SRC or DST | 00 | 00-11 | Register Direct | Rn | EA = Rn |
| | 01 | 00-11 | Register Indirect | (Rn) | EA = [Rn] |
| | 10 | vv | Absolute | Value | EA = Value |
| | 11* | vv | Immediate | #Value | Operand = Value |

EA = Effective Address
vv = Upper 2 bits of Value
* = SRC only

# Data Instructions

| OP | Fn | Assembly Language | Register Transfer Notation (RTN) |
|-----|------|-------------------|----------------------------------|
| 000 | MOVE | `MOVE SRC,DST` | DST ← SRC |
| 001 | ADD | `ADD  SRC,DST` | DST ← SRC + DST |
| 010 | INV | `INV  SRC,DST` | DST ← not SRC |
| 011 | AND | `AND  SRC,DST` | DST ← SRC and DST |
| 100 | ROTL | `ROTL SRC,DST` | DST ← SRC(14 dt 0) & SRC(15) |
| ... | | | |

# Branch Instruction Format

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
```

| OP | MD | OFFSET |  |
|----|----|--------|--|

IR

| OP | MD | Fn | Assy Lang | RTN |
|----|----|----|-----------|-----|
| 111 | 00 | BRA | `BRA  Offset` | PC ← PC + Offset |
|  | 01 | BGTZ | `BGTZ Offset` | PC ← PC + Offset  (STATUS > 0) |
|  | 10 | BSR | `BSR  Offset` | STACK ← PC; PC ← PC + Offset |
|  | 11 | RTN | `RTN` | PC ← STACK |

# FPGA Implementation
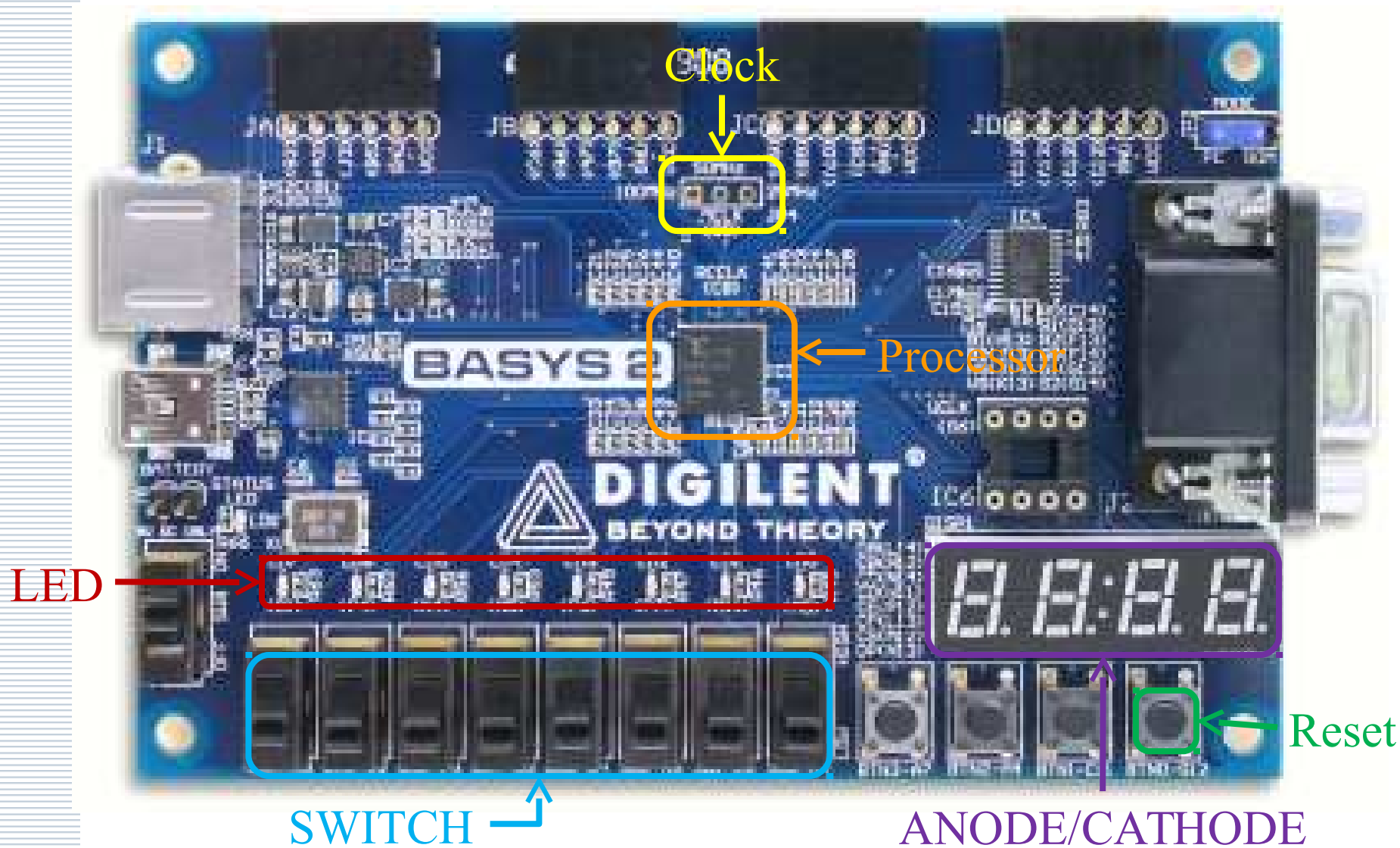
# Summary

- **Example Microprocessors**
  - Instruction Set Architecture
- **Instructional Processor Design**
  - Data Path
  - Memory
  - Instruction Processing
- **VHDL Model**
  - ISim Simulation
  - FPGA Implementation