UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering

**EEC180B** **DIGITAL SYSTEMS II** **Fall 1999**

# Lab 7-10: Micro-processor Design: Minimal Instruction Set Processor (MISP)

**Objective:** In this lab you will design, simulate and implement a small 16-bit microprocessor. You will exercise your elementary knowledge of computer organization and you will apply your digital design knowledge and elementary skills. The processor (MISP) is given to you by its architectural specifications. Those specifications are given to you in the MISP Architecture Document that follows. You will simulate and implement your design. Your design will be a pipelined processor containing a five cycle pipeline and capable of resolving data dependencies in hardware. As a final point, you are required to run an architectural verification program (AVP). This program will exercise all of the architected instructions and verify that your processor operates correctly. If your processor can pass the test - you have succeeded.

**Pre-lab:** You must show up to each of the lab sessions with your pre-lab completed. Otherwise, you will not be allowed to proceed with the lab. You have four weeks to complete this project. However, beware that if you do not start working immediately you will loose your precious time and ultimately fail. For pre-lab, do the complete paper design for the problem given below:

Lab 7. Complete a drawing of a block level organization of your computer. Identify all the registers that you need, multiplexers, major logic units, data paths and buses and all the control signals:

a. List them.
b. Provide the timing diagram for each of the instructions and show which control signals are activated and when for each of the instructions.
c. Provide the state-transition diagram and show clearly how will each instruction follow those states and in which phases.

Lab 8. Explain how are you going to implement pipelining and how are you going to resolve the data-dependencies. Draw a detailed diagram explaining the function and the structure of each component you are using.

a. Show all the possible conflicts between instructions (list them)
b. Show your solution for each one of the conflicts listed.
c. Show a detailed diagram of your data-path

Lab 9. Show the detailed diagram of your Control Unit.
a. Assign op-codes for your instructions. Explain advantage of your op-codes over the ones provided to you in this lab. How do they affect your control ?

Lab 10. Show your AVP and explain your verification strategy.

## I. <u>Minimal Instruction Set Processor (MISP) Design:</u>

## MISP Architecture Document

**General:**

MISP architecture is given by the Instruction Set Architecture (ISA) and the registers that are visible to the programmer.

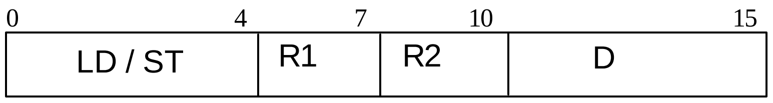
MISP is an 16-bit processor which data-path is 16-bits long and all of the instructions are register-to-register instructions 16-bit long. The architecturally visible register file contains 8 16-bit registers. Register R0 is wired to the value 0. This allows for fast comparisons and branches.

The data is treated as two's complement format representation for all the arithmetic operations. Arithmetic operation that involves Immediate field treats this 6-bit filed as 2's complement number. For logical operation the data is treated as 16-bits data.

**Instruction Formats:**

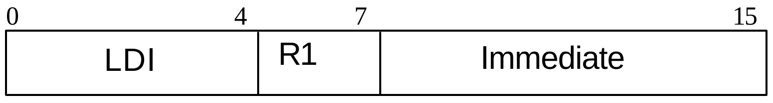
There are five instruction formats used in MISP. Opcode field is 5-bits long allowing for inclusion of 32 instructions in the ISA of MISP processor.

Load / Store:



Load and Store instruction uses Base + Displacement addressing mode. The destination register is specified by the field designated as R1. The base register is specified as R2 and this can be any register in the register file. Displacement field is carried with the instruction and it is 5-bits long allowing for a displacement of 31 entries from the beginning of a 32-entry page.
There are no flags set by this instructions and it is assumed that the data is always available.

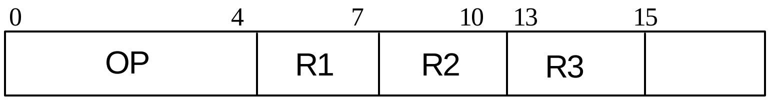
Load Immediate:

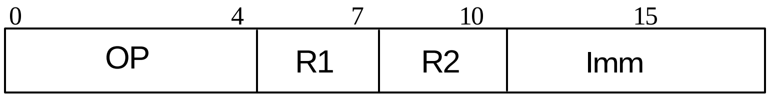


Load Immediate instructions loads the value specified in the 8-bit immediate field into the designated register. The value is treated as an unsigned integer.

There are no flags set by this instruction.

Arithmetic / Logic:





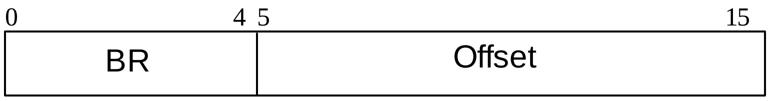
Arithmetic and logic instructions are of a RISC format, explicitly specifying the destination register R1, and two of the operand registers R2 and R3. The register field is 3-bits long allowing for each of the 8 registers to be addressed.

Another format used for Arithmetic operations only is Immediate format where a short 5-bit constant is carried in the instruction. Immediate field is treated as a 2's complement number. Therefore, the maximal number that can be directly coded into the instruction is 15 and the minimal -16.

Both Arithmetic instructions set the three condition bits in the condition code register:
N-negative, Z-zero result, Ov - overflow.

Logic instructions are using the three register format only. They do not set condition bits in the condition code register.

Branch:



Branch instruction tests the condition of one of the four condition bits of the condition register:

| | |
|---|---|
| N | negative |
| C | carry out (this is $C_{16}$ out of the ALU) |
| V | overflow bit (this is a logical XOR of $C_{16}$ and $C_{15}$) |
| Z | result equal to zero |

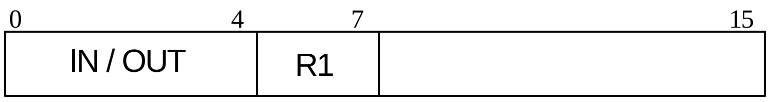
The condition bits are always set by the last instruction that modifies the condition codes.

If any one of the specified conditions is true the processor branches to the address specified as:

$$IAR \leftarrow IAR + Offset$$

Where IAR designates Instruction Address Register. The *Offset* field is 11-bits long which means that the processor can branch ahead for 1023, -1024 instructions from the current one. Therefore the branch is relative.

Input / Output:



Input and Output instruction transfer the data from IN and OUT peripheral register correspondingly. Input instruction (IN) transfer the content of the IN register to the general purpose register specified as R1. Output instruction transfer the content of the GPR, R1 to the OUT register. Those instructions are used to enable processor communications with the outside world.

**Table 1:** MISP Instruction Set

| **Load/ Store** | **Operation** |
|---|---|
| LD | R1 <- M[R2+D]; load R1 from memory location addressed as: A=Base+Displacement |
| ST | M[R1+D] <- R2; Store R2 to memory location addressed as: A=Base+Displacement |
| | |
| **Branch** | |
| BR: Always branch (unconditional) | Branch to the address relative to the current address for the distance Imm specified as a 2's complement number |
| BZ: Branch on Zero | if Z=1; IAR <- IAR + Offset; |
| BN: Branch on Negative | if N=1; IAR <- IAR + Offset; |
| BC: Branch on Carry | if C=1; IAR <- IAR + Offset; |
| BV: Branch on Overflow | if OV=1; IAR <- IAR + Offset; |
| | |
| **Arithmetic Operations:** | |
| ADD | R1 <- R2 + R3; add 2's complement |
| ADDI | R1 <- R2 + Imm; add immediate value |
| SUB | R1 <- R2 - R3; subtract 2's complement |
| NEG_A | R1 <- (-R2) ; negative 2's complement |
| INCR_A | R1 <- R2 + 1; increment |
| DEC_A | R1 <- R2 - 1; decrement |
| | |
| **Logic Operations** | |
| CMPL_A | $\overline{A}$ ; logical complement (invert all bits) |
| AND | R1 <- R2 AND R3; logical AND of all bits |
| OR | R1 <- R2 + R3; logical OR |
| XOR | $R_1 \leftarrow R_2 \oplus R_3$ ; exclusive OR |
| SHL_A | R1 <- 2R1 \|0; shift left one position. |
| | |
| **I/O Operations** | |
| IN | R1 <- IN; R1 receives the content of the input register IN |
| OUT | OUT <- R1; The value of the register R1 is placed in to the output register OUT |

**Implementation:**

The MISP processor is contains a General Purpose Register File (GPR File) containing 8 16-bit registers. The register file is three-ported allowing a simultaneous reading of the content of two register from the ports RA and RB as well as simultaneous writing through the port W. In case a read and write operation is performed in the same cycle and from the same register (Rx), GPR should be designed as a write-through. That means that the content written into the register Rx will be available on the read port RA (or RB) and latched into the staging register during the same cycle.

The processor is to be pipelined. The pipeline consists of the following five stages:

1. IF - Instruction Fetch
2. DEC - Instruction Decode
3. EX - Execute
4. MA - Memory Access
5. W - Write the result into the GPR

The data dependencies that occurs between the pipelined instructions is to be resolved in hardware. The by-pass buses as well as dependency comparators are to be implemented with the associated control logic so that the dependency resolution is performed directly in hardware and it is not known, nor visible, to the compiler.

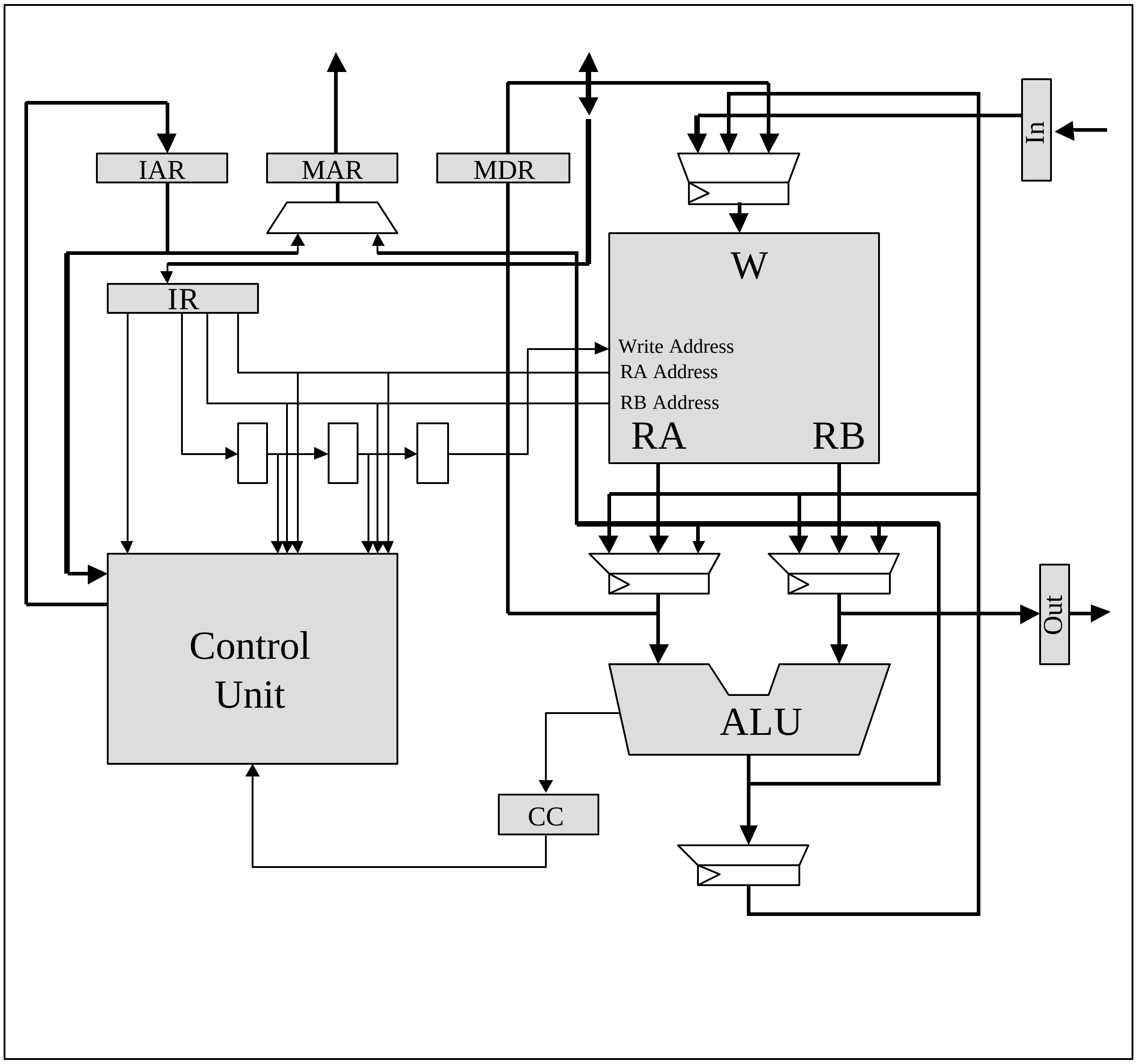


Fig. 1. General organization of the MISP processor (approximate)

Op-Code Assignments:

Enclosed is a suggested op-code assignment. However, you are free to modify the op-codes under a condition that you can provide a justification and explain how the newly assigned op-codes simplify your implementation.

**Op-Code Assignment Table:**

| **11** | **10** | **01** | **00** | **MSB / LSB** |
|---|---|---|---|---|
| ADD | CMPL_A | LD | NOP | **000** |
| ADDI | AND | ST | | **001** |
| SUB | OR | LDI | | **010** |
| NEG_A | XOR | | BV | **011** |
| INCR_A | SHL_A | | BZ | **100** |
| DEC_A | | | BN | **101** |
| | | IN | BC | **110** |
| | | OUT | BR | **111** |

## II. Implementing MISP processor in an Altera board

Based on the 180A Altera Tutorial II, you should be able to implement your circuit on an Altera board. You should target your design for an Altera EPF10K20 device. As in the tutorial lab, you should use one of the push-buttons for your Reset input. You should also use a clock divider circuit, as in the tutorial, in order to generate a reasonable clock frequency. (i.e. The frequency should allow you to see the output sequence clearly.) You can choose 4 of the 8 DIP-switches on the Altera board for your input switches. You can use Altera 7-segment display to show and demonstrate your operation.

## III. Lab Requirements

1. Design the ALU specified above using the Altera Max+Plus II CAD package.

2. Compile your circuit for a Flex 10K device. Verify your circuit by performing a timing simulation. Generate a printout of your simulation waveforms. Verify that the correct output sequence is

produced.

3. Download your design to an Altera board and verify the operation. Demonstrate your circuit to a TA.

## <u>IV. Lab Write-up</u>

Have your TA verify your timing simulation and then sign a verification sheet. For your lab report, include the following:

- ❑ Signed TA verification sheet.
- ❑ Graded pre-lab assignment
- ❑ Schematic of your circuit printed from Max+Plus II.
- ❑ Simulation waveforms produced by your timing simulation
- ❑ Description of your test set-up.