COE608: Computer Organization and Architectures
Winter 2017

Introduction to the Semi-RISC Processor

1. Introduction

This overview of the semi-RISC processing unit being implemented in the COE 608 course will be a useful reference throughout the course. Students may wish to refer it during the various laboratories where each laboratory is focused on building different parts (modules) of the processor.

2. CPU Specification

This document introduces the design features of the CPU; students will be exercising during the lab sessions of COE608.

- A 32-bit external data bus, 16-bit instruction bus, and 8-bit data memory bus. The 16-bit memory address enables 256B of instruction memory, whereas the custom made data memory supports 1KB.
- Two visible 32-bit working registers (A and B) used to store operands and results of data computation.
- The architecture is of Load/Store type. All data computations are performed in the working registers.
- Multi-cycle instructions along with a simple instruction set.
- Harvard based architecture with separate instruction and data storage.

The remainder of this document describes various components of the processor such as the instruction format, instruction set, register set and so forth. You will not concern with the interrupt/exception handling or stacks in this processor design.

2.1 Instruction Format

The instruction format is very simple. All instructions are 32-bits long and require a 4-byte memory word. For direct addressing mode, the 16-lower instruction bits represent an address location or a constant number. For storing and loading data memory, only the lower 8-bits are used for addressing. All other instructions are used for basic data processing and program flow control. The two instruction formats are illustrated in Figure 1.

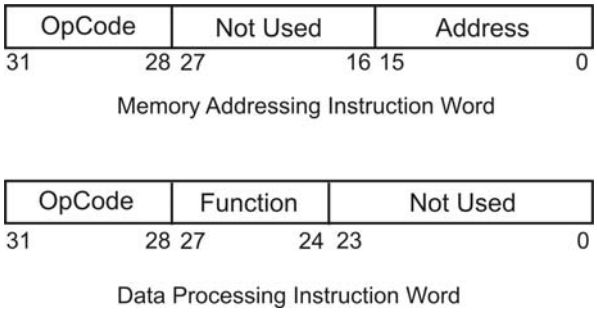

Figure 1: Instruction Format

2.2 Instruction Set

The instructions supported by the processor are listed in Table 1 including the instruction mnemonic, function and instruction word. The upper 4 instruction bits IR[31..28] indicate whether the operation is an addressing (or load constant) operation or a data processing operation (all data processing operations have an OpCode of "0111"). In the case of an addressing (ADDRS) operations and immediate (IMM) values, instruction bits IR[15..0] specify the exact value within the instruction. In the case of LUI operation a 16-bit constant at INST[15..0] is loaded into the upper 16-bits of register A. IMM based instructions such as LDAI, ADDI etc will load or propagate the instruction's immediate value into the appropriate registers and signals (i.e. LDAI: A<= IR[15:0]).

2.3 Register Set

The processor contains a number of registers used to perform various operations. Some of theses are accessible to the user while others are used to support internal operations. All user visible registers are illustrated in Figure 2

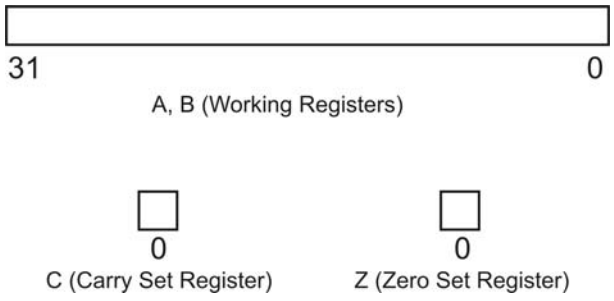

Figure 2: User Registers

Registers A and B are 32-bit working registers for data processing. Registers C and Z are 1-bit status registers used to indicate Carry and Zero detect, respectively (typically used in branching operations). The non-visible registers are used for internal operations and they are illustrated in Figure 3. The PC (Program Counter) register is used during instruction execution to specify the address of the next instruction. Register IR (Instruction Register) is used to hold instructions and for instruction decoding.

Table 1: Supported Instructions and their Format

Mnemonic	Function	Instruction Word		
		IR[31..28]	**IR[27..16]**	**IR[15..0]**
LDAI	A <= IR[15:0]	0000	X	IMM
LDBI	B <= IR[15:0]	0001	X	IMM
STA	M[ADDRS] <= A, ADDRS <= IR[15:0]	0010	X	ADDRS
STB	M[ADDRS] <= B, ADDRS <= IR[15:0]	0011	X	ADDRS
LDA	A <= M[ADDRS] , ADDRS <= IR[15:0]	1001	X	ADDRS
LDB	B <= M[ADDRS] , ADDRS <= IR[15:0]	1010	X	ADDRS
LUI	A[31:16] <= IR[15:0], A[15:0] <= 0	0100	X	IMM
JMP	PC <= IR[15..0]	0101	X	ADDRS
BEQ	IF(A==B) then PC <= IR[15..0]	0110	X	ADDRS
BNE	IF(A!=B) then PC <= IR[15..0]	1000	X	ADDRS
		IR[31..28]	**IR[27..24]**	**IR[15..0]**
ADD	A <= A + B	0111	0000	X
ADDI	A <= A + IR[15..0]	0111	0001	IMM
SUB	A <= A - B	0111	0010	X
INCA	A <= A + 1	0111	0011	X
ROL	A <= A << 1	0111	0100	X
CLRA	A <= 0	0111	0101	X
CLRB	B <= 0	0111	0110	X
CLRC	C <= 0	0111	0111	X
CLRZ	Z <= 0	0111	1000	X
ANDI	A <= A AND IR[15..0]	0111	1001	IMM
TSTZ	If Z = 1 then PC <= PC + 1	0111	1010	X
AND	A <= A AND B	0111	1011	X
TSTC	If C = 1 then PC <= PC + 1	0111	1100	X
ORI	A <= A OR IR[15..0]	0111	1101	IMM
DECA	A <= A - 1	0111	1110	X
ROR	A <= A >> 1	0111	1111	X

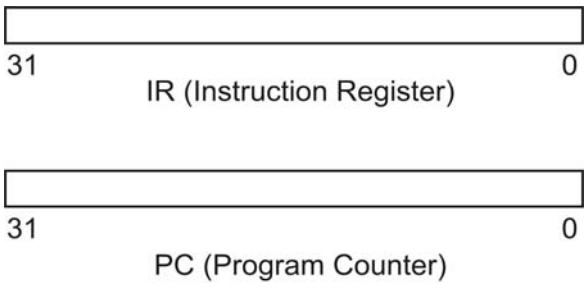

Figure 3: Internal Registers

3. Instruction Execution

Figure 4 shows an ASM chart for performing each instruction in the instruction set. Each instruction requires 3 clock cycles (denoted as T0, T1, T2 stages) to be completed. These clock cycles correspond to the following operations:

T0: Instruction Fetch 1. The address of the next program instruction is located in the PC. The content of the instruction at address PC is transferred to the IR.

T1: Instruction Fetch 2 and Memory Pre-decode. The PC is incremented to point at the next instruction. Pre-decoding and signal assertions are needed for Load and Store instructions. This is necessary to account for the setup times needed by the data memory.

- For LDA/ LDB, the en and wen signals must be set here accordingly to start loading the contents of A/B from data memory so that it may be latched in T2.
- For STA/ STB, the wen and en control signals and MUXES must be asserted in this stage for the setup time needed for the address and data to be stored by T2.

Note: Ensure that the PC is also simultaneously incremented while pre-decoding the memory instructions. Refer to Figure 5 for further details on the data memory enable states.

T2: Instruction Decode & Execute. The instruction is decoded here by examining the contents of IR. IR[31..28] bits will indicate the type of instruction (address versus data). IR[15..0] are also transferred to RED, PC or LZE in preparation for an load/store, addressing, or load immediate operation respectively. The actual execution of the operation is also performed here.

For LDA/LDB, ensure that the proper MUX and register signals are asserted to latch the data coming from memory. For STA/STB, make sure that the MUX signals continue to be asserted for correct hold times for storing to memory.

The above instruction sequence repeats indefinitely.

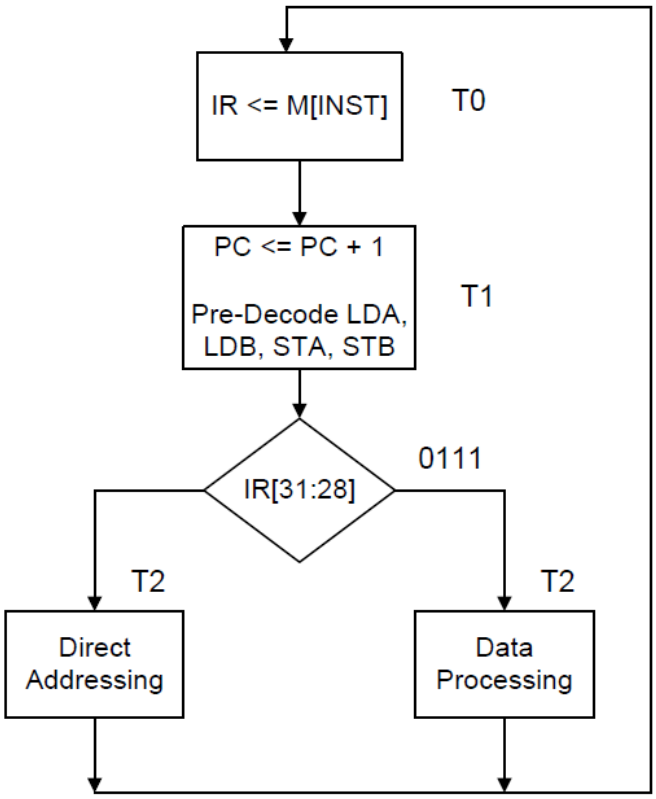

Figure 4: Instruction Execution ASM (State) Chart

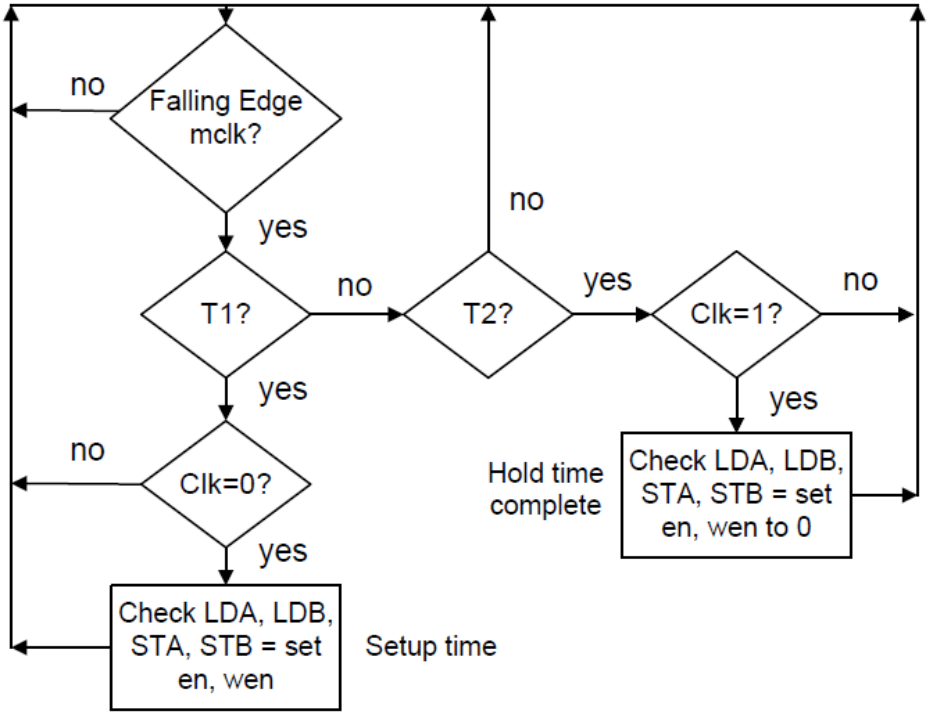

Figure 5: Data Memory Enable States ASM Chart