Advanced Computer Architecture

Lecture No. 17

Reading Material

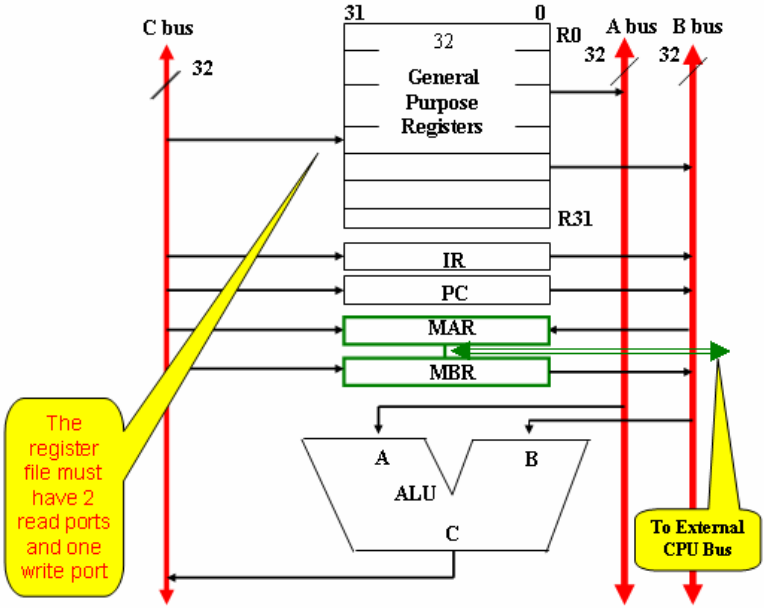
Vincent P. Heuring&Harry F. Jordan — Chapter 4
Computer Systems Design and Architecture — 4.6.2, 4.7, 4.8

Summary

- 3-bus implementation for the SRC
- The Machine Reset
- Machine Exceptions

A 3-bus Implementation for the SRC

Let us now look at a 3-bus implementation of the data-path for the SRC as shown in the figure. Two buses, 'A' and 'B' bus for reading, and a bus 'C' for writing, are part of this implementation. Hence all the special purpose as well as the general purpose registers have two read ports and one write port.



Structural RTL for the Subtract Instruction using the 3-bus Data Path Implementation

We now consider how instructions are fetched and executed in 3-bus architecture. For this purpose, the same 'sub' instruction example is followed.

The syntax of the subtract instructions is

sub ra, rb, rc

The structural RTL for implementing this instruction is given in the table. We observe that in this table, only two time steps are required for the instruction fetch phase. At time step T0, the Memory Address Register receives the value of the Program Counter. This is done in the initial phase of the time step T0. Then, the Memory Buffer Register

receives the memory word indexed by the MAR, and the PC value is incremented. At time step T1, the instruction register is assigned the instruction word that was loaded into the MBR in the previous time step. This concludes the instruction fetch and now the instruction execution can commence.

	Step	RTL
Instruction Fetch	T0	MAR←PC; MBR ← M[MAR], PC ← PC + 4;
	T1	IR ← MBR;
Instruction Execute	T2	R[ra] ← R[rb] - R[rc];

In the next time step, T2, the instruction is executed by subtracting the values of register rc from rb, and assigning the result to the register ra.
At the end of each sequence, the timing step generator is initialized to T0

Control Signals for the Fetch Operation
The given table lists the control signals in the instruction fetch phase. The control signals for the execute phase can be written in a similar fashion.

Step	RTL	Control Signals
T0	MAR←PC; MBR ← M[MAR], PC ← PC + 4;	PCout, INC4, LPC, LbMAR, MRead,
T1	IR←MBR;	MBRout, C=B, LIR;
T2	Instruction_ Execution	

The Machine Reset

In this section, we will discuss the following
- Reset operation
- Behavioral RTL for SRC reset
- Structural RTL for SRC reset

The reset operation
Reset operation is required to change the processor's state to a known, defined value. The two essential features of a reset instruction are clearing the control step counter and reloading the PC to a predefined value. The control step counter is set to zero so that operation is restarted from the instruction fetch phase of the next instruction. The PC is reloaded with a predefined value usually to execute a specific recovery or initializing program.
In most implementations the reset instruction also clears the interrupt enable flags so as to disable interrupts during the initialization operation. If a condition code register is present, the reset instruction usually clears it, so as to clear any effects of previously executed instructions. The external flags and processor state registers are usually cleared too.

The reset instruction is mainly used for debugging purposes, as most processors halt operations immediately or within a few cycles of receiving the reset instruction. The processors state may then be examined in its halted state.
Some processors have two types of reset operations. Soft reset implies initializing PC and interrupt flags. Hard reset initializes other processor state registers in addition to PC and interrupts enable flags. The software reset instruction asserts the external reset pin of the processor.

Reset operation in SRC

Hard Reset
The SRC should perform a hard reset upon receiving a start (Strt) signal. This initializes the PC and the general registers.
Soft Reset
The SRC should perform a soft reset upon receiving a reset (rst) signal. The soft reset results in initialization of PC only.
The reset signal in SRC is assumed to be external and asynchronous.
PC Initialization
There are basically two approaches to initialize a PC.
1. **Direct Approach**
The PC is loaded with the address of the startup routine upon resetting.
2. **Indirect Approach**
The PC is initialized with the address where the address of the startup routine is located. The reset instruction loads the PC with the address of a jump instruction. The jump instruction in turn contains the address of the required routine.
An example of a reset operation is found in the 8086 processor. Upon receiving the reset instruction the 8086 initializes its PC with the address FFFF0H. This memory location contains a jump instruction to the bootstrap loader program. This program provides the system initialization

Behavioral RTL for SRC Reset
The original behavioral RTL for SRC without any reset operation is:
Instruction_Fetch :=(! Run&Strt: (Run ← 1; instruction_Fetch,
Run : (IR ← M [PC]; PC ← PC+4;instruction_execution)),
instruction_execution:= (ld (:=op=1…) ;
This recursive definition implies that each instruction at the address supplied by PC is executed. The modified RTL after adding the reset capability is
Instruction_Fetch:=(! Run&Strt :(Run ← 1,
PC, R [0...31] ← 0),
Run&!Rst :(IR ← M [PC],
PC ← PC+4, instruction_execution);
Run&Rst:(Rst ← 0, PC ← 0);
instruction_Fetch),
The modified definition includes testing the value of the “rst” signal after execution of each instruction. The processor may not be halted in the midst of an instruction *in the RTL definition*
To actually implement these changes in the SRC, the following modification are required to add the reset operation to the structural RTL for SRC:

- A check for the reset signal on each clock cycle
- A control signal for clearing the PC
- A control signal to load zero to control step counter

Example: The sub instruction with RESET processing
To actually reset the processor in the midst of an instruction, the "Rst" condition must be tested after each clock cycle.

Step	RTN	Control Sequence
T0	!Rst:(MA ← PC,C ← PC+4), Rst:(Rst ← 0,PC ← 0,T ← 0)	!Rst:(PC_{out}, LMAR, INC4, LC, MRead), Rst:(ClrPC, Goto0);
T1	!Rst:(MD ← M[MA]:PC ← C), Rst:(Rst ← 0:PC ← 0:T ← 0)	!Rst:(C_{out}, LPC, Wait), Rst :(ClrPC, Goto0);
T2	!Rst:(IR ← MD), Rst:(Rst ← 0: PC ← 0:T ← 0)	!Rst:(MBR_{out}, LIR), Rst : (ClrPC, Goto0);
T3	!Rst:(A ← R[rb]), Rst:(Rst ← 0: PC ← 0: T ← 0)	!Rst:(RBE, R2BUS, LA), Rst : (ClrPC, Goto0);
T4	!Rst:(C ← A - R[rc]), Rst:(Rst ← 0: PC ← 0:T ← 0)	!Rst:(RCE, R2BUS, SUB, LC), Rst :(ClrPC, Goto0);
T5	!Rst:(R[ra] ← C), Rst:(Rst ← 0:PC ← 0: T ← 0)	!Rst:(LC, RAE, BUS2R, End), Rst : (ClrPC, Goto0);

Let us examine the contents of each phase in the given table. In step T0, if the Rst signal is not asserted, the address of the new instruction is delivered to memory and the value of PC is incremented by 4 and stored in another register. If the "Rst" signal is asserted, the "Rst" signal is immediately cleared, the PC is cleared to zero and T, the step counter is also set to zero. This behavior (in case of 'Rst' assertion) is the same for all steps. In step T1, if the rst signal is not asserted, the value stored at the delivered memory word is stored in the memory data register and the PC is set to its incremented value.
In step T2, the stored memory data is transferred to the instruction register.
In step T3, the register operand values are read.
In step T4, the mathematical operation is executed.
In step T5, the calculated value is written back to register file.
During all these steps if the Rst signal is asserted, the value of PC is set to 0 and the value of the step counter is also set to zero.

Machine Exceptions

- Anything that interrupts the normal flow of execution of instructions in the processor is called an exception.
- Exceptions may be generated by an external or internal event such as a mouse click or an attempt to divide by zero etc.
- External exceptions or interrupts are generally asynchronous (do not depend on the system clock) while internal exceptions are synchronous (paced by internal clock)

The exception process allows instruction flow to be modified, in response to internal or external events or anomalies. The normal sequence of execution is interrupted when an exception is thrown.

Exception Processing

A generalized exception handler should include the following mechanisms:

1. **Logic to resolve priority conflicts**. In case of nested exceptions or an exception occurring while another is being handled the processor must be able to decide which exception bears the higher priority so as to handle it first. For example, an exception raised by a timer interrupt might have a higher priority than keyboard input.
2. **Identification of interrupting device**. The processor must be able to identify the interrupting device that it can to load the appropriate exception handler routine. There are two basic approaches for managing this identification: exception vectors and "information" register. The exception vector contains the address of the exception handling routine. The interrupting process fills the exception vector as soon as the interruption is acknowledged. The disadvantage of this approach is that a lot of space may be taken up by vectors and exception handler codes.
   In the information register, only one general purpose exception handler is used. The PC is saved and the address of the general purpose register is loaded into the PC. The interrupting process must fill the information register with information to allow identification of the cause and type of exception.
3. **Saving the processor state**. As stated earlier the processor state must be saved before jumping to the exception handler routine. The state includes the current value of the PC, general purpose registers, condition vector and external flags.
4. **Exception disabling during critical operation**. The processor must disable interrupts while it is switching context from the interrupted process to the interrupting process, so that another exception might not disrupt the transition.

Examples of Exceptions

- Reset Exception
  Reset operation is treated as an exception by some machines e.g. SPARC and MC68000.
- Machine Check
  This is an external exception caused by memory failure
- Data Access Exception
  This exception is generated by memory management unit to protect against illegal accesses.
- Instruction Access Exception
  Similar to data access exception

- Alignment Exception
  Generated to block misaligned data access

Types of Exception

- **Program Exceptions**
  These are exceptions raised during the process of decoding and executing the instruction. Examples are illegal instruction, raised in response to executing an instruction which does not belong to the instruction set. Another example would be the privileged instruction exception.
- **Hardware Exceptions**
  There are various kinds of hardware exceptions. An example would be of a timer which raises an exception when it has counted down to zero.
- **Trace and debugging Exceptions**
  Variable trace and debugging is a tricky task. An easy approach to make it possible is through the use of traps. The exception handler which would be called after each instruction execution allows examination of the program variables.
- **Nonmaskable Exceptions**
  These are high priority exceptions reserved for events with catastrophic consequences such as power loss. These exceptions cannot be suppressed by the processor under any condition. In case of a power loss the processor might try to save the system state to the hard drive, or alert an alternate power supply.
- **Interrupts (External Exceptions)**
  Exception handlers may be written for external interrupts, thus allowing programs to respond to external events such as keyboard or mouse events.