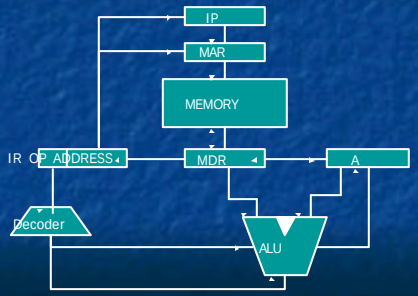
Lecture Notes

01/12/04

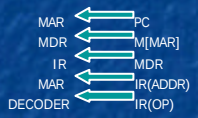
By: Derek Kelley

Timothy Crofton

Von-Neumann Machine



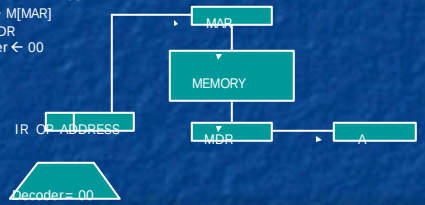
00 FETCH Cycle



The Fetch Cycle is a set of "hidden" instructions that the programmer does not need to worry about.

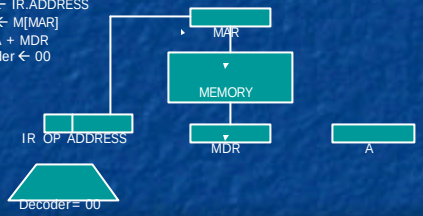
01 Load

FETCH
MAR ← IR.ADDRESS
MDR ← M[MAR]
A ← MDR
Decoder ← 00



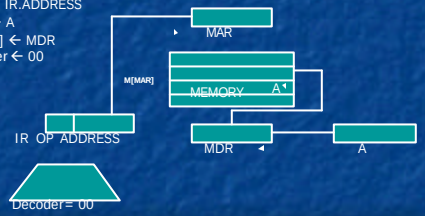
02 ADD

FETCH
MAR ← IR.ADDRESS
MDR ← M[MAR]
A ← A + MDR
Decoder ← 00



03 STORE

FETCH
MAR ← IR.ADDRESS
MDR ← A
M[MAR] ← MDR
Decoder ← 00



Error Handling

At this point, we have assumed we have no errors in our single address machine.

However, we need to consider possible overflow errors.
An example using four bit addition:

```
  1 1 1 1
  1 0 0 0
---------
1 0 1 1 1
```

Since only 4 bits are allowed, how must we handle this 5th bit?

Overflow

The addition of a flip/flop to our ALU would enable us to check for this overflow situation.

The CPU has to check for an interrupt each time an instruction is executed.

Therefore, we need to add a new command to the ISA, the Interrupt Cycle (05 ABEND).

At the end of each execution cycle, the DECODER will be set to 05 instead of 00, to check for interrupts at the end of each execution cycle.

Halting at this step enables us to prevent inconsistent data.

ISA Interrupt Cycle

01 LOAD
MAR ← IR.ADDRESS
MDR ← M[MAR]
A ← MDR
DECODER ← 05

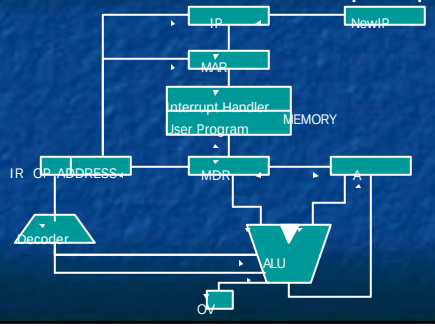
02 ADD
MAR ← IR.ADDRESS
MDR ← M[MAR]
A ← A + MDR
DECODER ← 05

03 STORE
MAR ← IR.ADDRESS
MDR ← A
M[MAR] ← MDR
DECODER ← 05

04 END

05 Int. Handler Routine
IF OV = 1 then HALT
DECODER ← 00

VN with Overflow Flip/Flop



Interrupt Handler

Instead of stopping the program when an overflow is encountered, the flow of execution is simply passed to an Interrupt Handler.

This also allows us to take advantage of pipelining since the program is not halted.

To do this, the program counter (IP) is loaded with the start address of the interrupt handler in memory from NEWIP.

05 ABEND
IF OV = 1 then IP ← NEWIP
DECODER ← 00

Virtual Machine

The Interrupt Handler is the first extension layer of the "Virtual Machine".

This is the first step towards an operating system.

Shared Memory

Like any other program, the Interrupt Handler must be loaded into memory.

However, this may lead to a new problem if the currently loaded program tries to modify the interrupt handler's routine.

We need some way to protect the interrupt handler's routine from being modified by the currently loaded program.

Memory Protection

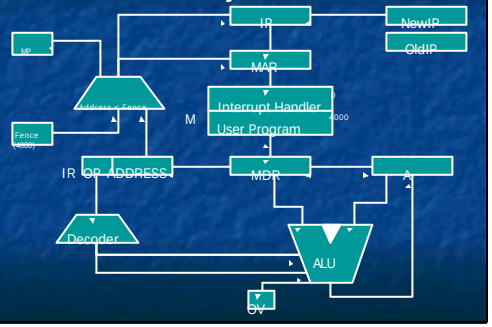
We protect the interrupt handler's routine by adding three components:

Fence Register- Register loaded with the address of the boundary between the interrupt handler routine and the user program.

Device for Address Comparison – compares the fence register with any addresses that the user program attempts to access.

Flip/Flop – The flip/flop is set to 1 if a memory violation occurs.

VN with Memory Protection



Updated ISA

01 LOAD
MAR ← IR.ADDRESS
IF MP = 0 then
  MDR ← M[MAR]
  A ← MDR
DECODER ← 05

02 ADD
MAR ← IR.ADDRESS
IF MP = 0 then
  MDR ← M[MAR]
  A ← A + MDR
DECODER ← 05

03 STORE
MAR ← IR.ADDRESS
IF MP = 0 then
  MDR ← A
  M[MAR] ← MDR
DECODER ← 05

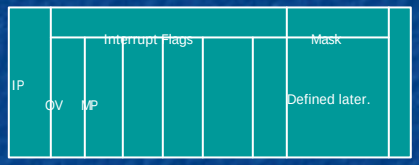
05 Int. Handler Routine
IF OV = 1 IP ← NEWIP
IF MP = 1 IP ← NEWIP
DECODER ← 00

Program State Word(PSW)

The Program State Word (PSW) is a device that gives us information about the program's current state.

In this register we have the IP, MODE, Interrupt Flags, and Mask (to be defined later).

Program State Word(PSW)



Privileged Instruction

What would happen if a user program tried to modify a register that shouldn't be changed, such as the FENCE register?

Clearly we don't want to change the FENCE value, however, the FENCE register is not protect under the memory protection mechanism.

By using the idea of privileged instructions we are able to deno te which instructions are allowed by the user program and which one s are prohibited (system only instructions).

Privileged Instructions

The computer needs a way to distinguish when privileged instructions are allowed and when they aren't.

This can be accomplished by operating in two distinct modes.

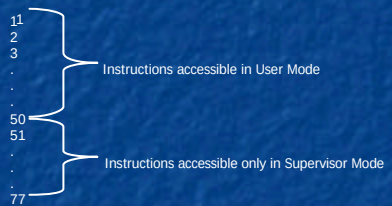
User Mode – 0

Supervisor Mode– 1

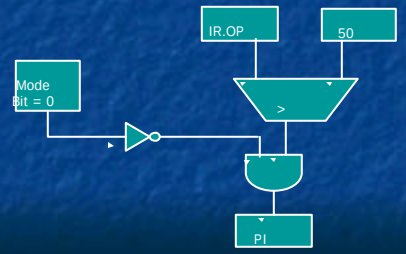
When in user mode, the program is restricted to limited set of instructions.

When in the CPU is in supervisor mode, all instruction are avail able.

Privileged Instruction Set



User/Supervisor Implementation



05 Interrupt Cycle

If OV = 1 THEN IP ← NEWIP; MODE ← 1 (ABEND).
If MP = 1 THEN IP ← NEWIP; MODE ← 1 (ABEND).
If PI = 1 THEN IP ← NEWIP; MODE ← 1 (ABEND).
DECODER ← 00

Types of Interrupts

Software Interrupts

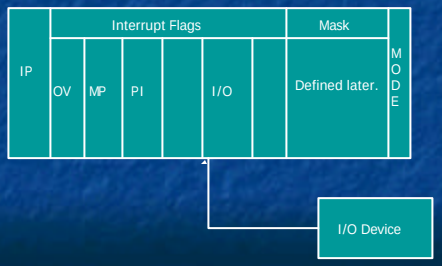
Traps – Overflow Problem

System Calls – The program tells the O.S. to do something for it.

Hardware (I/O) Interrupt – When the operation assigned to the O.S. is completed, a signal is sent.

External (Timer)– Implemented to protect against an infinite loop.

Program State Word



05 Interrupt Cycle

If OV = 1 THEN IP ← NEWIP; MODE ← 1 (ABEND).
If MP = 1 THEN IP ← NEWIP; MODE ← 1 (ABEND).
If PI = 1 THEN IP ← NEWIP; MODE ← 1 (ABEND).
If I/O = 1 THEN OLDIP ← IP
IP ← NEWIP
MODE ← 1
DECODER ← 00

Timer Interrupts

A device is needed to prevent a program from "hogging" the CPU.

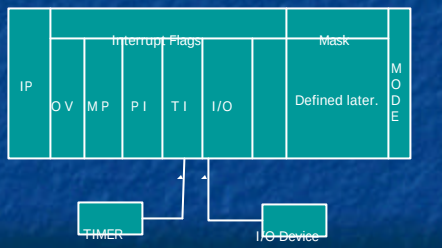
This is done by setting a timer once a program starts. If the program has not completed by the time the timer gets to zero, the Timer Interrupt (TI) bit is set to 1.

This transfers control to the Interrupt Handler.

05 Interrupt Cycle

If OV = 1 THEN IP ← NEWIP; MODE ← 1 (ABEND).
If MP = 1 THEN IP ← NEWIP; MODE ← 1 (ABEND).
If PI = 1 THEN IP ← NEWIP; MODE ← 1 (ABEND).
If I/O = 1 THEN OLDIP ← IP
IP ← NEWIP
MODE ← 1
If TI = 1 THEN OLDIP ← IP
IP ← NEWIP
MODE ← 1
DECODER ← 00

Program State Word



SuperVisorCall (SVC)

An SVC is also known as a System Call.

It is a mechanism to request service from the Supervisor or OS.

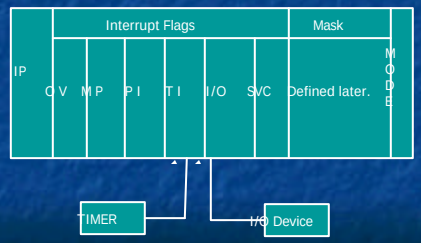
This mechanism is a type of interrupt, called a *software interrupt* because the program itself relinquishes control to the Supervisor as part of its instructions.

System Call Example

From the previous slide

1. Allows user programs to ask for service (instructions found below opcode 50)
2. Privileged Instructions (over opcode 50)

Program State Word



More was covered on SVC in the next lecture…