Zagazig University **Computer & Systems Engineering Dept.**

Faculty of Engineering

Z80 – SIMULATOR
INTERRUPTS AND INTERRUPT HANDLING

OBECTIVES

1. To introduce the concept of interrupts as a means to perform Input/Output operations.
2. To show the different types of interrupts usually available in microprocessors.
3. To demonstrate how interrupts are handled.
4. To be familiar with Z80- Simulator software.

EQUIPMENT REQUIRED

1. PC
2. Z80- Simulator

INTRODUCTION TO INTERRUPTS

In simple applications we conceive systems in which the microcomputer program dominates the sequence of events. The reading of inputs, sending of outputs etc is all done as the appropriate codes are executed. It is easy to see that the real world does not always fit into this ideal situation.

For example, if an input represents a dangerous condition that suddenly occurs then the microprocessor will not detect it until the program get round to reading it. A possible solution to this is to regularly check the input and respond if necessary. This is called POLLING.

Unfortunately, if an input is critical, it is easy to get to a situation where the system checks for this input every few microseconds (just in case!) but the input only occurs once every six months or so!

Interrupts represent a much more suitable alternative. An interrupt line is a connection direct to the microprocessor. A signal on this line is a request for the microprocessor to stop whatever it is doing at the time and so something else that is considered to be more important. On completion of this the original task would, generally, be resumed.

You must understand how different this is to our previous ideas. With interrupts, something OUTSIDE the system can DEMAND action but it only does so when necessary. This means that under normal conditions the microprocessor has the maximum amount of time available to get on with the job in hand.

The Z80 microprocessor has two Interrupt lines called -NMI and –INT (The “-“implies 'active low').

-NMI: Stands for NON MASKABLE INTERRUPT. The line is normally at logic 1 but if this falls to logic 0 then the falling edge will interrupt normal program execution. The programmer cannot prevent this and –NMI is normally used to detect power failures, safety hazards etc.

-INT: Stand for INTERRUPT REQUEST. This too is normally held at logic 1 but becomes active if it is sent to logic 0. In this case however the programmer is able to determine whether the event actually leads to suspension of current operations or whether it is ignored. This is determined by the state of an INTERRUPT ENABLE FLIP FLOP (IFFI) in the microprocessor. The Z80 and its family of input/output circuits provide a comprehensive interrupt system based in these two external signals, and a number of special instructions are provided to manage it.

Overview

At the most elementary level the concept of any interrupt operation is as follows.

The Z80 checks on the state of the two interrupt lines on completion of each instruction cycle. If either of them is active (and in the case of -INT, is enabled) the Z80 ceases to execute the normal program and instead executes another program called an INTERRUPT SERVICE ROUTINE. This program performs whatever tasks are required when an interrupt is detected. On completion of this service routine control is (normally) returned to the original program that was being executed when the interrupt occurred.

Obviously the idea raises all sorts of questions. How is normal execution resumed? How is the source of interrupt identified if several external devices can produce -INT or -NMI signals? What happens if an interrupt occurs when one is being serviced and so on.

To answer these, and other important questions, we begin with the case of **Non-Maskable Interrupts**.

Non-Maskable Interrupts.

There are three sources of an -NMI signal.

The most obvious is a signal applied to the –NMI socket that is available on the MIC960 interface connection. This is provided for the USER.

The action that will produce an -NMI here is the FALLING EDGE of a 5V to 0V transition.

However the NMI connection can also be pulled low in this way by signals within the system. Single step execution or pressing the HALT key both produce NMI's.

When an -NMI occurs from any source the Z80 responds as follows.

1. It **disables** the **-INT** line
2. The next instruction in the 'normal' program is ignored and instead the program counter is loaded with 0066H. The 'normal' program counter value is saved in a special area of RAM called the **STACK**.

Having loaded the program counter with 0066H, execution begins at this point which is part of the monitor program. The monitor program executed is an NMI SERVICE ROUTINE and its prime function is as follows

1. To determine whether the -NMI was caused by a USER SIGNAL or by a SYSTEM SIGNAL (due to HALT or single step etc).
2. To carry out whatever is required by a system interrupt, or, alternatively, to allow USER interrupts to be acted upon. This last aspect is the most important to the user and to employ it the following is required.
    1. The USER must provide an –NMI service routine to meet his own needs.
    2. The STAR ADDRESS of this service routine must be placed in a special pair of memory locations. These are called USER NMI vectors.

Placing this start address in predetermined memory locations allows the monitors -NMI service routine to pass control to your service routine in the event of an active signal applied to -NMI socket (pin).

The starting address of the NMI service routine is **0066H** on the simulator memory.

The USER -NMI Service Routine.

Basically this does whatever you require in response to the –NMI signal but note the following.

1. Although the Z80 has saved a program counter value to allow execution of the main program to resume later, it saves nothing else. This means that *the contents of any register or address used by the interrupt service routine are corrupted as far as the main program is concerned*. As such some means of protecting or saving the values is usually necessary. One simple way is to switch to the ALTERNATE REGISTER SET for interrupt service operations.

2. To return from the -NMI service to the original program the service MUST end with the instruction, **RETN**. This instruction will restore the original state of the INT enable/disable system (which was disabled when NMI occurred) and will then retrieve the original program counter value from the stack.

   Once this is returned to the program counter normal execution resumes.

Interrupt Requests (-INT).

The general concepts that we have investigated with -NMI also apply to -INT though there are three main differences.

1. INT responds to a level (logic 0) not an edge.
2. INT can be disabled.
3. There are three different methods of application. These are called MODES.

The ability to enable or disable any response to -INT going low is handled by two instructions **EI** and **DI** that control a flip flop IFF1, The INTERRUPT ENABLE FLIP FLOP. The **EI** instruction, 'ENABLE INTERRUPTS', sets IFF1 and thus enables Interrupt requests. The **DI** instruction, 'DISABLE INTERRUPTS', clears IFF1 and thus disables interrupt requests.

An additional flip flop IFF2 is used to keep track of the value in IFF1 so that when an -NMI automatically disables the -INT system by clearing IFF1, then IFF1 can subsequently be restored to its original state by reference to IFF2 which is unaffected by the -NMI.

To use any form of Interrupt Request the instruction EI must have been issued first because INT is initially disabled during power up.

Interrupt Modes.

The three modes are known as MODE 0, 1 and 2.

The mode to be used is chosen by executing the instructions IM0, **IM1** or IM2 respectively. For this introductory curriculum only Mode 1 will be used. In mode 1 the concept of operation is just like that of NMI with an

Interrupt handling routine in the monitor checking for 'user' signals on the -INT socket and passing control to the user if this is the case.

The starting address of the NMI service routine is **0038H** on the memory simulator.

Although we will not apply modes 0 or 2 here their concept is relevant. With mode1, any signal that pulls -INT low externally will cause execution to move to the vectored address just as -NMI did. However, in many 'real' applications there may be numerous sources of the -INT signals all capable of pulling the Z80's INT pin low. Modes 0 and 2 allow various means of discriminating between different sources by accepting an external data input and using this as a means of indicating what produced the -INT signal. Action is then taken accordingly.

PROCEDURE

Part 1-Maskable Interrupts :

1. launch Z80 – Simulator from start menu → programs → Z80 Simulator IDE.
2. Tools → Assembler, an untitled file will be opened, save it with the name Z80INT1.asm, then enter program 1, BUT note:

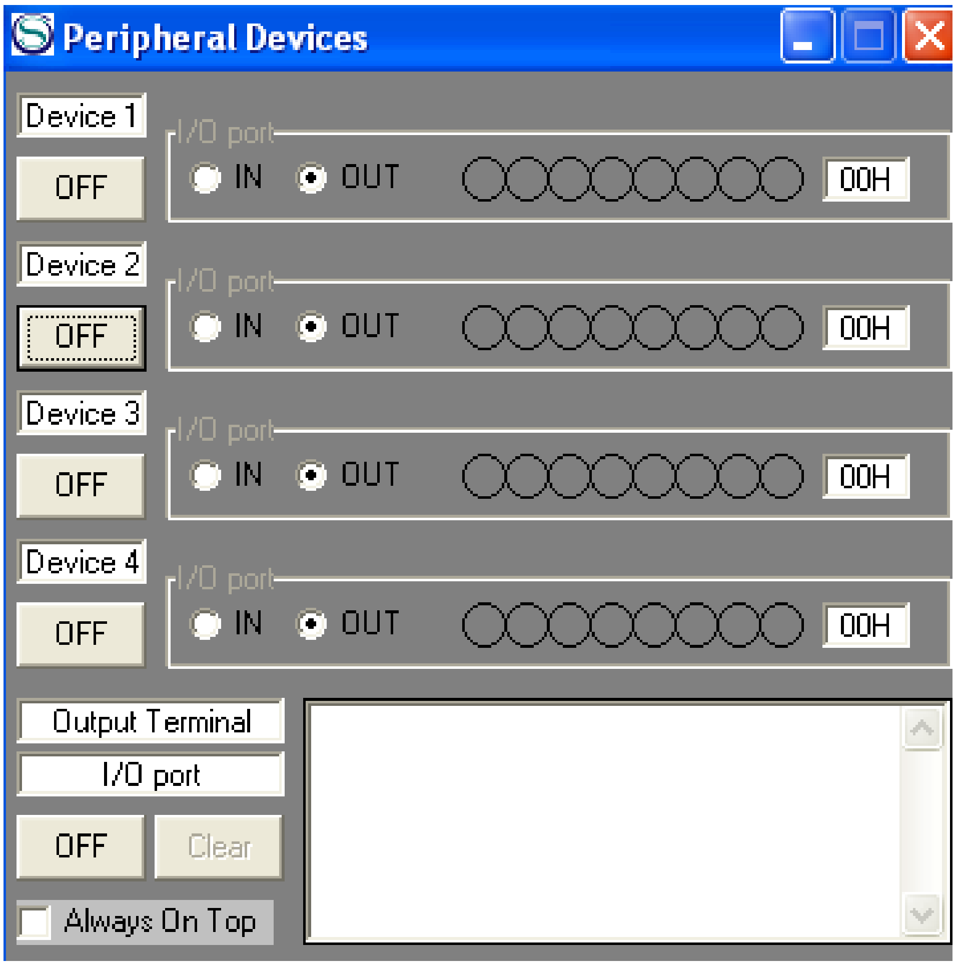
   *Write the statements as written follows **& keep the spaces** carefully.*

Program 1

JP 5000H	
; INT ISR	comment
.ORG 0038H	Starting address of ISR
EXX	Exchange the registers B , C&D With B ' ,C' & D'
LD D, 055H	
LD C, 03H	
OUT (C), D	Out D contents on Port 03H

CALL WAIT	Delay
LD D, 00H	Clear port 03H
OUT (C), D	
EXX	
RETI	Return to Main program
; MAIN PROG.	comment
.ORG 5000H	Starting address in memory
IM 1	Mode 1 of maskable interrupt
EI	Enable the maskable interrupts
LD C, 02H	
LD D, 01H	
LOOP: OUT (C), D	Out D contents on Port 02
RRC D	Rotate right contents of D
CALL WAIT	
JP LOOP	
.END	End of the program
WAIT: LD B, 01H	Delay subroutine
LP1: LD A, 0FH	
LP2: DEC A	
JR NZ, LP2	
DJNZ LP1	Decrement B and jump relative if B=0.
RET	

3. from **the window of the assembler** choose **Tools → Assemble**
4. If there are errors, correct them, then Tools → Load.
5. Thus your program is assembled & loaded into the memory.
6. from **the main window of Z80** simulator choose Tools → Memory Editor
7. Enter locations 38H & 5000H and **Comment**.
8. Choose Rate → Fast.
9. Choose Tools → Peripheral Devices:



- Click on OFF button of Device1 and enter 02, OK.
- Click on OFF button of Device2 and enter 03, OK.

10. Choose Simulator → start , to start executing your program.
11. Note the output of Port 02, **Comment**
12. Click on **INT** button in the main window, Note Port 02 & port 03 outputs,
13. And the contents of PC register, **Comment.**
14. Note that after returning from the ISR, INT button becomes disabled
15. **What can you do to enable it again?**
16. on finishing, from Simulator → stop.

Hint: RESET button starts the program again.

Part 2. Non--Maskable Interrupts:

1. Load program 1 but replace the following instructions:
   .ORG 0038H with **.ORG 0066H**
   RETI with **RETN**
2. Run the new program, then click on NMI button, **Comment**
3. Remove the **RETN** instruction, Run the program and click on NMI button

What is happen? Why?

Report :

Answer all the **questions** & write all the required **comments** in the steps of part 1 & part 2.

Z80 – Instruction Set

ADC	HL,ss	Add with carry register pair ss to HL.
ADC	A,s	Add with carry operand s to accumulator.
ADD	A,n	Add value n to accumulator
ADD	A,r	Add register r to accumulator.
ADD	A,(HL)	Add location (HL) to acccumulator.
ADD	A,(IX+d)	Add location (IX+d) to accumulator.
ADD	A,(IY+d)	Add location (IY+d) to accumulator.
ADD	HL,ss	Add register pair ss to HL.
ADD	IX,pp	Add register pair pp to IX.
ADD	IY,rr	Add register pair rr to IY.
AND	s	Logical AND of operand s to accumulator.
BIT	b,(HL)	Test bit b of location (HL).
BIT	b,(IX+d)	Test bit b of location (IX+d).
BIT	b,(IY+d)	Test bit b of location (IY+d).
BIT	b,r	Test bit b of register r.
CALL	cc,nn	Call subroutine at location nn if condition CC is true.
CCF		Complement carry flag.
CP	s	Compare operand s with accumulator.
CPD		Comapre location (HL) and acc., decrement HL and BC,
CPDR	.	Perform a CPD and repeat until BC=0
CPI		Compare location (HL) and acc., incr HL, decr BC.
CPIR		Perform a CPI and repeat until BC=0.
CPL		Complement accumulator (1's complement).
DAA		Decimal adjust accumulator.
DEC	m	Decrement operand m.
DEC	IX	Decrement IX.
DEC	IY	Decrement IY.
DEC	ss	Decrement register pair ss.
DI		Disable interrupts.
DJNZ	e	Decrement B and jump relative if B=0.
EI		Enable interrupts.
EX	(SP),HL	Exchange the location (SP) and HL.
EX	(SP),IX	Exchange the location (SP) and IX.
EX	(SP),IY	Exchange the location (SP) and IY.
EX	AF,AF'	Exchange the contents of AF and AF'.
EX	DE,HL	Exchange the contents of DE and HL.
EXX		Exchange the contents of BC,DE,HL with BC',DE',HL'.
HALT	.	Halt computer and wait for interrupt
IM	0	Set interrupt mode 0.
IM	1	Set interrupt mode 1.
IM	2	Set interrupt mode 2.
IN	A,(n)	Load the accumulator with input from device n.
IN	r,(c)	Load the register r with input from device (C).
INC	(HL)	Increment location (HL).
INC	IX	Increment IX.
INC	(IX+d)	Increment location (IX+d).
INC	IY	Increment IY.
INC	(IY+d)	Increment location (IY+d).

INC	r	Increment register r.
INC	ss	Increment register pair ss.
IND		(HL)=Input from port (C). Decrement HL and B.
INDR		Perform an IND and repeat until B=0.
INI		(HL)=Input from port (C). HL=HL+1. B=B-1.
INIR		Perform an INI and repeat until B=0.
JP	(HL)	Unconditional jump to location (HL).
JP	(IX)	Unconditional jump to location (IX).
JP	(IY)	Unconditional jump to location (IY).
JP	cc,nn	Jump to location nn if condition cc is true.
JR	C,e	Jump relative to PC+e if carry=1.
JR	e	Unconditional jump relative to PC+e.
JR	NC,e	Jump relative to PC+e if carry=0.
JR	NZ,e	Jump relative to PC+e if non zero (Z=0).
JR	Z,e	Jump relative to PC+e if zero (Z=1).
LD	A,(BC)	Load accumulator with location (BC).
LD	A,(DE)	Load accumulator with location (DE).
LD	A,I	Load accumulator with I.
LD	A,(nn)	Load accumulator with location nn.
LD	A,R	Load accumulator with R.
LD	(BC),A	Load location (BC) with accumulator.
LD	(DE),A	Load location (DE) with accumulator.
LD	(HL),A	Load location (HL) with accumulator.
LD	dd,nn	Load register pair dd with nn.
LD	dd,(nn)	Load register pair dd with location (nn).
LD	HL,(nn)	Load HL with location (nn).
LD	(HL),r	Load location (HL) with register r.
LD	I,A	Load I with accumulator.
LD	IX,nn	Load IX with value nn.
LD	IX,(nn)	Load IX with location (nn).
LD	(IX+d),n	Load location (IX+d) with n.
LD	(IX+d),r	Load location (IX+d) with register r.
LD	IY,nn	Load IY with value nn.
LD	IY,(nn)	Load IY with location (nn).
LD	(IY+d),n	Load location (IY+d) with value n.
LD	(IY+d),r	Load location (IY+d) with register r.
LD	(nn),A	Load location (nn) with accumulator.
LD	(nn),dd	Load location (nn) with register pair dd.
LD	(nn),HL	Load location (nn) with HL.
LD	(nn),IX	Load location (nn) with IX.
LD	(nn),IY	Load location (nn) with IY.
LD	R,A	Load R with accumulator.
LD	r,(HL)	Load register r with location (HL).
LD	r,(IX+d)	Load register r with location (IX+d).
LD	r,(IY+d)	Load register r with location (IY+d).
LD	r,n	Load register r with value n.
LD	r,r'	Load register r with register r'.
LD	SP,HL	Load SP with HL.
LD	SP,IX	Load SP with IX.
LD	SP,IY	Load SP with IY.
LDD		Load location (DE) with location (HL), decrement DE,HL,BC.
LDDR		Perform an LDD and repeat until BC=0.
LDI		Load location (DE) with location (HL), incr DE,HL; decr BC.
LDIR		Perform an LDI and repeat until BC=0.
NEG		Negate accumulator (2's complement).
NOP		No operation.

OR	s	Logical OR of operand s and accumulator.
OTDR		Perform an OUTD and repeat until B=0.
OTIR		Perform an OTI and repeat until B=0.
OUT	(C),r	Load output port (C) with register r.
OUT	(n),A	Load output port (n) with accumulator.
OUTD		Load output port (C) with (HL), decrement HL and B.
OUTI		Load output port (C) with (HL), incr HL, decr B.
POP	IX	Load IX with top of stack.
POP	IY	Load IY with top of stack.
POP	qq	Load register pair qq with top of stack.
PUSH	IX	Load IX onto stack.
PUSH	IY	Load IY onto stack.
PUSH	qq	Load register pair qq onto stack.
RES	b,m	Reset bit b of operand m.
RET		Return from subroutine.
RET	cc	Return from subroutine if condition cc is true.
RETI		Return from interrupt.
RETN		Return from non-maskable interrupt.
RL	m	Rotate left through operand m.
RLA		Rotate left accumulator through carry.
RLC	(HL)	Rotate location (HL) left circular.
RLC	(IX+d)	Rotate location (IX+d) left circular.
RLC	(IY+d)	Rotate location (IY+d) left circular.
RLC	r	Rotate register r left circular.
RLCA		Rotate left circular accumulator.
RLD		Rotate digit left and right between accumulator and (HL).
RR	m	Rotate right through carry operand m.
RRA		Rotate right accumulator through carry.
RRC	m	Rotate operand m right circular.
RRCA		Rotate right circular accumulator.
RRD		Rotate digit right and left between accumulator and (HL).
RST	p	Restart to location p.
SBC	A,s	Subtract operand s from accumulator with carry.
SBC	HL,ss	Subtract register pair ss from HL with carry.
SCF		Set carry flag (C=1).
SET	b,(HL)	Set bit b of location (HL).
SET	b,(IX+d)	Set bit b of location (IX+d).
SET	b,(IY+d)	Set bit b of location (IY+d).
SET	b,R	Set bit b of register r.
SLA	m	Shift operand m left arithmetic.
SRA	m	Shift operand m right arithmetic.
SRL	m	Shift operand m right logical.
SUB	s	Subtract operand s from accumulator.
XOR	s	Exclusive OR operand s and accumulator.