Test 3 Study Guide ECE362

Austin Sypolt

* List step by step actions of 68HC12 after an interrupt occurs

First the CPU halts continuation of the current program, loads the address of the beginning address of ISR (interrupt service routine) into program counter and executes the ISR, upon completion of the interrupt the CPU returns and continues the main program.

The CPU finishes up the currently executed instruction, saves the return address to the program counter, saves other registers onto the stack, then jumps to the ISR and executes it. Upon executing RTI, the CPU restores all registers and then returns to the main program and continues.

* Explain how to enable maskable interrupts? How to disable them?

By default maskable interrupts are globally disabled however they can be enabled by setting the I flag of the CCR to one. On reset the I flag is set (all disabled by default).

* What is the main difference between software interrupt (SWI) and calling a subroutine?

A subroutine is a portion of code within a larger program. It performs a specific task and is typically independent of the remaining code. Interrupt Service Routines (ISRs) are used to handled hardware interrupts. These routines are not independent threads, but more like signals. ISR is called if any thread is suspended by an interrupt.

A subroutine runs when you call it. An ISR runs whenever a certain signal occurs (this signal can be generated by software or hardware). The big difference is that you know where the subroutine runs because you call it, however you do not know when the ISR will be executed as its an already preset section of code.

* Write an RTI interrupt service routine with the name RTI\_SER. The interrupt service routine should be able to send an 8-bit counter to port\_S (LEDs) to indicate how many RTI interrupts have occurred. You do not need to write the main program for interrupt initialization and port\_S initialization.
* You need to test if your RTI interrupt service routine written above works. Write a complete S12 main program to test the above RTI\_SER routine. Don’t forget to initialize the vector table.
* Draw 68HC12 address/data demultiplexing circuit for the normal expanded-narrow mode.
* Program for converting 8-bit binary to BCD.

HW7

1. What is a 4-bit BCD value of 10111010 (base 2)?

Convert to decimal, divide by 10^n (of highest n) and change the direct division value of that to binary then put them together.

1. What is the difference between polling and interrupt? Use an example to describe the operation of polling and interrupt.

In polling, the processor is tied up waiting for an event to occur. In an interrupt environment, the expected event signals the processor when it occurs, allowing the processor to handle other tasks in the meantime.

1. How many different interrupts does 68HC12 have?

Total about 25, they are initialized in our vector table and can be called from there. This consists of both maskable and nonmaskable interrupts.

Interrupts contained in 68HC12:

* Unimplemented Instruction Trap (UM)
* Software Interrupt (UM)
* XIRG (UM)
* IRQ (M)
* Real Time Interrupts (M)
* Timer Channels (M)
* Timer Overflow (M)
* Pulse Accumulator Overflow (M)
* Pulse Accumulator Edge Detect (M)
* Serial Peripheral Interface (M)
* Serial communications Interface (M)
* Analog-to-Digital (M)
* Key Wake-up (M)

1. What is a maskable interrupt?

A type of interrupt that the programmer can disable.

1. What is a nonmaskable interrupt?

A type of interrupt that the programmer cannot disable as they are vital to the operation and maintenance of the CPU.

1. What is interrupt service routine (ISR)?

A software routine to be executed in response to an interrupt occurred.

The ISR is a special block of code associated with specific interrupt conditions that will be performed when properly flagged/called.

1. What is the relationship between reset and interrupt?

Reset is a type of nonmaskable interrupt.

1. What is the interrupt vector table? How is it used by the CPU? How to initialize the interrupt vector table?

The interrupt vector table is a predefined space for storing interrupt vectors. Used by the CPU to redirect it to the beginning of the ISR with the properly called interrupt. Initialization of the table is done using a linker file and placing the ICR addresses, which are numerically numbered vector values.

1. How to enable and disable maskable interrupts? How to enable and disable non-maskable interrupts?

To enable and disable a maskable interrupt the I flag of the CCR is raised, letting the CPU know that an interrupt has occurred. Use assembly instructions cli or sei or manipulate the I flag using bit manipulation.

Non-maskable interrupts are always enabled as they are vital to the operation of the CPU.

1. How many different interrupt service routines can 68HC12 have?

The same number as the number of interrupts/types. (From the vector table)

1. How does the CPU know which interrupt service routine to run after getting an interrupt?

The CPU first determines which interrupt was received and then gets the address of the corresponding ISR from the interrupt vector from the interrupt vector table.

1. List step-by-step actions of the CPU when an interrupt occurs.

* The CPU finishes execution of the current instruction
* The starting address of the proper ISR from the interrupt vector table is found
* CPU saves the return address and all registers automatically.
* CPU executes the ISR.
* CPU restores all registers and returns the return value upon execution of RTI.

1. Can you call an interrupt service routine in your main program without crashing the program?

No, generally you run interrupts as subprograms and not part of your main program, this is due to interrupts having a variety of uses and applications, so they should be as independent and as straight to the point as possible.

1. What is a software interrupt (SWI)? What is the difference between a software interrupt and calling a subroutine?

A software interrupt is an interrupt triggered by the SWI assembly instruction in the program. Maskable interrupts are disabled during the ISR.

In a subroutine the registers are not automatically saved and restored upon completion, as well as the I flag not being used in a subroutine. A subroutine is also typically in the middle of code for more specific processes.

1. Give an example of using the software interrupt.

Testing the interrupt setup.

Testing the interrupt service routines.

Debug monitoring by setting breakpoints in the execution of your code.

1. What is interrupt priority? When is interrupt priority useful?

Interrupt priority is the hierarchical relationship between the interrupts that decides which interrupt to execute first. This is useful in instances where multiple interrupts are queued, and one may have higher priority so the one with higher priority will be executed first and then upon completion the next queued up ISR will initiate.

When multiple interrupts occur at the same time, interrupt priority determines which interrupt will be served first.

1. Can a higher priority-interrupt interrupt the CPU while it is executing a lower priority ISR.

Yes, this is possible if the higher priority interrupt is non-maskable as well as if the higher priority interrupt is maskable and the lower priority ISR enables the interrupt.

However, not in the case where the higher priority interrupt is maskable and the lower priority ISR disables further interrupts.

1. Can a lower priority-interrupt interrupt CPU while it is executing a higher priority ISR?

No, if the lower priority-interrupt is non-maskable.

Yes, if the higher priority interrupt is maskable and the I flag has been cleared in its ISR.

1. Is it appropriate to include a long program code in the ISR?

No. A long program in the ISR decreases the responsiveness of the system so ISR’s typically are as small as possible to save space and operation time.

1. What is a nested interrupt? Does 68HC12 support nested interrupts?

A nested interrupt is to allow interrupts while executing another ISR.

Our 68HC12 boards do support nested interrupts.

1. Write a program that defines an 8-bit global variable COUNTER and initializes the IRQ interrupt vector in the interrupt vector table. The name of the ISR is IRQ\_ISR. It should send the value of COUNTER every 10ms to an output port at address $D6.

XDEF COUNTER ; if necessary

XREF IRQ\_ISR ; if necessary

My\_code: Section

Entry:

sei

Ldd #IRQ\_ISR

Std $fff2 ;enable the device interrupt

cli

repeat: ldx #10000

delay10ms: dex

bne delay10ms

movb COUNTER,$D6

bra repeat

b) Write an ISR that counts the number of times itself is executed.

XREF COUNTER

XDEF IRQ\_ISR

My\_code: Section

IRQ\_ISR:

Inc COUNTER

…

…

…

RTI

HW8

1. What are the three operation modes on the HC12 microcontrollers? For each mode specify the sizes of external address and data buses.

Single chip mode: No external buses

Expanded narrow mode: 16 bit address bus, 8 bit data bus

Expanded wide mode: 16 bit address bus, 16 bit data bus

1. What kind of circuit do you need in order to extract address signals from a CPU with multiplexed address and data signals?

Address latch or address/data demultiplexing circuit.

1. Why should a bidirectional buffer be used when connecting a CPU to the data bus?

To be able to isolate the CPU from the data bus and to amplify the current. Also it is used to control the data flow direction.

1. Why is the latch optional in the following circuit? In what situation is it essential and in what situation is it optional?

A8-A15 is not multiplexed with data pins so address/data demultiplexing circuit is not needed for A8-A15. It is optional if you want the A0-A7 and A8-A15 to have the same timing characteristics.

If A8-A15 and D8-D15 share the same pins, address/data demultiplexing circuit is essential for A8-A15.

(So it is dependent on whether or not you’re using a 8 or 16-bit data bus).

1. Explain the following timing diagram

This timing diagram is the relationship of all signals used for memory and I/O interface. We know that the memory and CPU are connected through the address bus, data bus and control bus. Therefore, in this diagram we expect to see address information, data information, and read/write control information. We also expect to see clock information (ECLK) as this is the standard system that drives our CPU. The read/write control signal is R/W. The address and data information are represented twice, there is an upper one for reading operation and a lower one for writing operation.

The falling edge of ECLK starts the memory reading/writing cycle. The rising edge of the ECLK tells the latch that the AD pins contain the valid address and should be latched. For a memory read cycle, the time (23) indicated within how long the memory should provide valid data back. Time (11) and (12) show how long the memory chips hold data valid on the data bus. (3) shows when the CPU will read the data from the data bus after providing a valid address. The R/W control signal stays high all the time.

For a memory write cycle, time (13) indicates how long after providing a valid address the CPU will provide valid data. Time (14) and (15) show how long the CPU will hold data valid on the data bus. The R/W control signal goes to low for (16) after the start of memory write.

All signals are provided by the CPU except that the data is provided by the CPU during write operations and provided by the memory during read operations.

ECE362 Test 3 Study Guide

Austin Sypolt

* What is a key advantage of using interrupts over other ways?

Interrupts remain dormant until called allowing for that time to be used for other processes, this is the downside of polling.

* Explain how to enable and disable maskable interrupts.

Maskable interrupts can be enabled by setting the I flag in the CCR through bit manipulation or by calling the interrupt using a command like sei or cli, they are automatically disabled upon completion of the ISR.

* Why is it critical to keep interrupt service routines as short (small) as possible?

This is a necessity because interrupts are called all the time and should take up the smallest amount of space and take the smallest amount of time to execute due to it stalling other system processes.

* Why and when do we need to have interrupt priority mechanism?

When multiple interrupts occur at the same time, an interrupt priority mechanism determines which interrupt will be served first.

When some interrupts are more important (urgent) to serve, while executing an ISR, an interrupt priority mechanism needs to determine if it allows further interrupts.

* List step by step actions of 68HC12 after an interrupt occurs.

The 68HC12 finishes it current instruction

Disables interrupts and saves the return address and all registers

Jumps to correct ISR

Executes ISR

Upon RTI, using the saved return address, it jumps back to the process that was interrupted, and restores all registers.

* In what cases, can a higher-priority-interrupt interrupt the CPU while it is executing a lower priority ISR?

This is possible in the cases where a lower priority interrupt is used in the execution of the higher priority interrupt.

If a high priority interrupt is nonmaskable.

If the high priority interrupt is maskable and the low priority ISR enables maskable interrupts by clearing the I flag.

* An interrupt occurred while executing a single byte instruction at memory location $1234. The CPU is executing its corresponding interrupt service routine. Right before the interrupt, the contents of SP were $89AF. What is the value of SP right before CPU executes the last instruction in the interrupt service routine.

$89A6

$89AF-8 = $89F7 – 1 = $89F6

* What is the main difference between a software interrupt and calling a subroutine?

Registers are saved automatically or manually in a SWI, while a subroutine does not do that.

Further interrupts are automatically disabled during the SW ISR.

An interrupt is a jump to a predefined section of code defining the ISR, while a subroutine is a jump to a section typically in your own code that is more specific to your code.

* Assume that a lower priority maskable interrupt occurred while an S12 CPU is executing the interrupting service routine of a higher priority maskable interrupt. Describe precisely when the lower priority interrupt will be serviced in two cases.

1. If further interrupts are enabled (in a higher priority ISR), the lower priority ISR will execute immediately after the execution of the current instruction.
2. If further interrupts are disabled (in a higher priority ISR), the lower priority will execute upon the completion of the higher priority ISR.



