Interrupts and exceptions

Lecture Topics

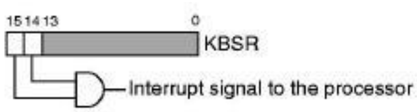
- Interrupt-driven I/O
- Interrupts and exceptions

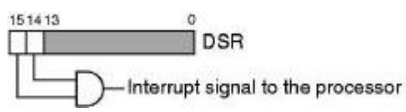
Lecture materials

- Textbook §8.5, §10.2

Interrupt-driven I/O

- In memory-mapped I/O, interaction with the I/O devices is controlled by our program
  - Our program polls ready bits of I/O registers to see if the I/O devices are ready for interaction
  - This leads to inefficiencies since our program effectively stalls until an I/O operation is complete
- In case of interrupt-driven I/O, interaction with the I/O device is controlled by the I/O device itself
  - An I/O device generates an interrupt signal to indicate that I/O device is ready with new I/O operation (e.g., a new character has been entered on the keyboard)
  - In response to this interrupt, the currently executed program stops its execution and the control is passed to some subroutine designed to handle the interrupt
  - Once the subroutine processes the interrupt, the control is passed back to the program that was previously executed
- Several things must be true for an I/O device to actually interrupt the processor:
  - The I/O device must want to request service
    - This is indicated by ready bit (KBSR[15] and DSR[15]). If these bits are set, there is a new I/O request ready to be served
  - The device must have the right to request service
    - This is indicated by an **interrupt enable bit** (KBSR[14] and DSR[14]). If such bit is set by the processor, the processor wants to give the I/O device the right to request the interrupt service
    - ready bit and interrupt enable bit together are used to generate an interrupt



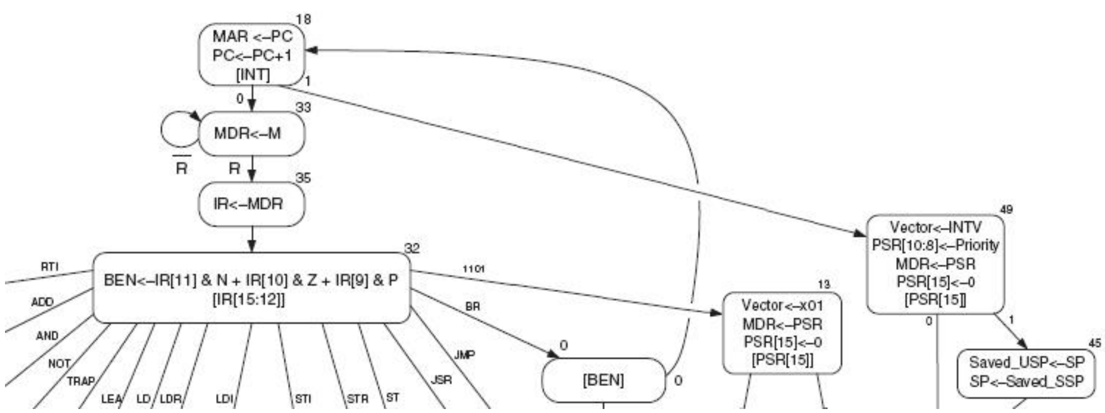


  - This request must be more urgent than the processor's current task
    - A program is executed with some specified priority level, LC-3 has 8 such priority levels PL0..PL7.
- How interrupts work
  - Basic flow
    - Stop the running program on any instruction
    - Vector to some other piece of code
    - Resume right where we left off
  - Some other capabilities we may want to have
    - Nest interrupts (have an interrupt be interrupted)
    - Turn off interrupts for a period of time
    - Block interrupts of lower priority (needed for deadlock avoidance)
    - Prevent user programs from executing certain instructions or assessing certain state
- Interrupts vs. Exceptions
  - Interrupts are due to some outside influences beyond the currently running program
    - Examples: I/O, timer interrupt, etc.
  - Exceptions are caused by the currently running program
    - Examples: illegal instructions, protection violation, etc.

- Interrupt Handling
    - Events: Hit key on a keyboard, execute illegal opcode, disk ready to transfer data, etc.
    - Current user program must be suspended to process these urgent requests by an interrupt handler
    - At the software level, handling an interrupt is like calling a subroutine, only that the software state at the time of the call is less structured (there is no user control over when interrupts occur in comparison with subroutine calls in program)
        - Save PC so system knows where to return when interrupt service is done
        - Save all registers so that they can be used, and restore them when interrupt service is done
        - Save the condition codes (NZP) because they are set and tested in different instructions
- LC3 Interrupt Handling
    - Not all interrupts are created equal (PL0-PL7)
        - LC3 maintains an interrupt priority (PSR[10:8])
        - Devices wanting to interrupt have a 3-bit priority
    - When interrupt happens
        - Device asserts the interrupt request signal (INT) and presents an 8-bit interrupt vector (INTV)
            - Used to construct a memory address that contains the location of the interrupt handler in a jump table
    - Process interrupts in supervisory mode (PSR[15]=0)
        - Processing in supervisory mode uses a different stack pointer (Supervisory Stack Pointer, or SSP) than in user mode
        - Information saved onto supervisory stack before interrupts are processed
            - USP (User Stack Pointer)
            - PSR[15] (supervisory mode), PSR[10:8] (current priority mode), PSR[2:0] (condition code NZP)
            - PC - 1 (decrement PC because PC points to instruction past the one subverted)
- Interrupt Registries and Control Signals. We need
    - Interrupt vector registry (INTV)
    - Priority registry
    - Processor status register (PSR)
    - Memory for pointers (user stack, supervisor stack)
    - Temporary storage for PC and PSR
    - Circuits to generate and handle interrupt signals
- LC3 Interrupt Table
    - Each device is associated with an 8-bit vector to index an interrupt vector table
    - Interrupt vector table is in memory
        - Between x0100 and x01FF
        - Each contains beginning address of service routine for handling interrupt
    - Exception service routines (x0100-x017F)
        - Handle exception events that prevent program from executing correctly
    - Interrupt service routines (x0180-x01FF)
        - Handle service events external to running program
- I/O Interrupt Handling

- Only interrupt from keyboard in LC3
    - Priority level PL4 (out of 8 levels)
    - 8-bit interrupt vector (INTV=x80 located at x0180)
- Assumptions of the I/O interrupt
    - A program is running at priority level less than 4
    - Interrupt Enable is set (IE=1) for Keyboard Status Register (KBSR) when key is pressed

- Procedure for I/O Interrupt Handling
    - **IF** program is running at priority < PL4 AND IE(KBSR)=1 **AND** someone strikes a key on a keyboard then
        - Set Supervisory mode (PSR[15]=0)
        - Set Priority to PL4 (PSR[10:8] = 100)
        - R6 <- Supervisory Stack Pointer (SSP)
        - Push Processor Status Register (PSR) and PC of interrupted program to Supervisor Stack
        - Expand 8-bit interrupt vector (x80) from keyboard to x0180 (address to interrupt table)
        - Load PC with value stored at memory address x0180
- Exception Handling
    - Only Exceptions from
        - Privilege mode violation
            - If processor encounter RTI when in User mode
        - Illegal opcode
            - If IR[15:12] == 1101 is true (unused opcode = 1101)
    - Exception handling
        - Similar to interrupt handling
- Procedure for Exception Handling
    - **IF** processor encounter RTI when in User mode **OR** IR[15:12] == 1101 then
        - Set Supervisory mode (PSR[15]=0)
        - R6 <- Supervisory Stack Pointer (SSP)
        - Push Processor Status Register (PSR) and PC of interrupted program to Supervisor Stack
        - Expand 8-bit interrupt vector (x00 OR x01) to (x0100 OR x0101) the address to interrupt vector table
        - Load PC with value stored at memory address x0100 OR x0101
- Checking for Interrupts



- State 18 is the only state in which the processor checks for interrupts (before 'begin fetch' phase)
  - Best to check for interrupts before a new instruction is executed
- In State 18, Check for INT
  - If INT=0 (no interrupt) go to State 33
    - Next state (NS) = 100001 (33)
  - If INT=1 go to State 49 (110001)

- Interrupt Micro-Instruction

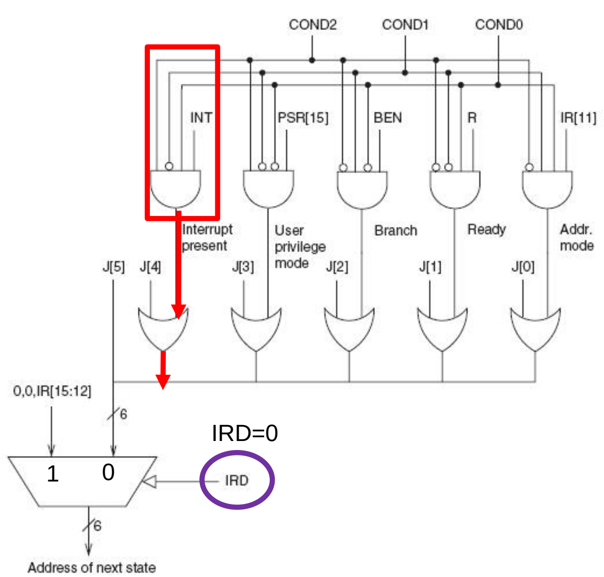
Control Address 18

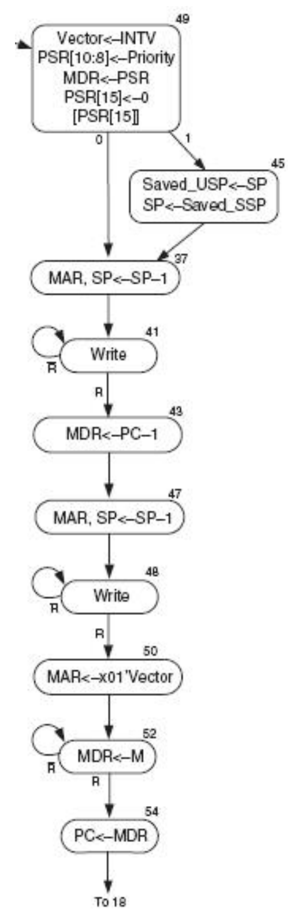
48	47	46	45	44	43	42	41	40	39	38	0
0	1	0	1	1	0	0	0	0	1		

IRD **COND** **J** **CONTROL SIGNALS**

If INT=1 then NS = 110001 (49)

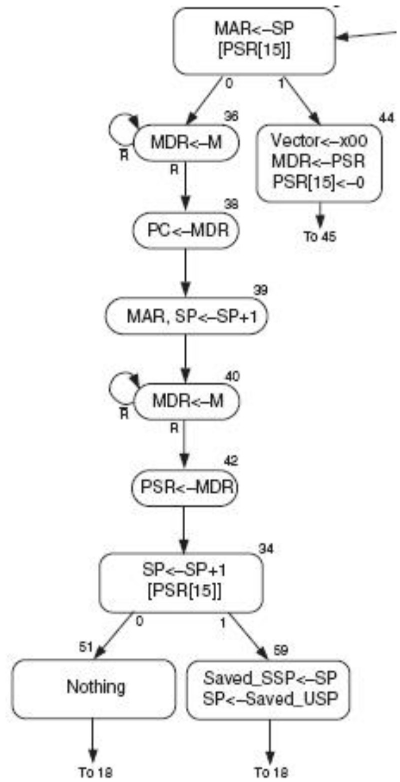
- State 18: Micro-sequencer Control
  - COND=101 ; Test for interrupts J[4]
  - J=100001 ; Default next state = 33
  - J=110001 ; Otherwise, next state = 49



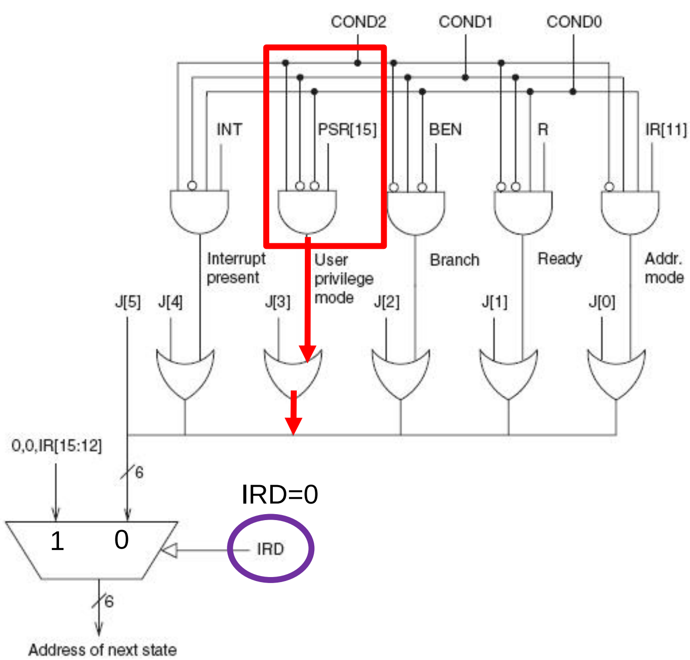


- Processing an Interrupt
  - Load PSR (with privilege mode, priority level, and condition code of interrupt program) to MDR, in preparation for pushing into Supervisory Stack
  - Record Priority Level and INTV provided by interrupting device
  - Test old PSR[15]
    - If old PSR[15] = 1 then system was in User mode and hence save USP (R6) in Saved_USP, load R6 with Saved_SSP, go to state 37
    - If old PSR[15] =0 then system was in supervisory mode already
  - Save PSR, old PC to Supervisory Stack
  - Load PC with address of interrupt service routine

- Returning from Interrupt (RTI)
  - Restore PSR and PC
  - If PSR[15] == 0 then RTI continues
    - Restore PC first
    - Restore PSR next
    - Test old PSR[15]
      - If old PSR[15] = 1 then the system returns to User mode and hence restore USP (R6) and store SSP
      - If old PSR[15] = 0 then system continues to be in the Supervisory mode
  - If PSR[15] = 1 (Privilege Mode Exception)
    - Handle condition as an privileged mode violation
    - Load Interrupt Vector with starting address of Privilege mode violation
    - Go to State 45 to handle interrupt as if by INT



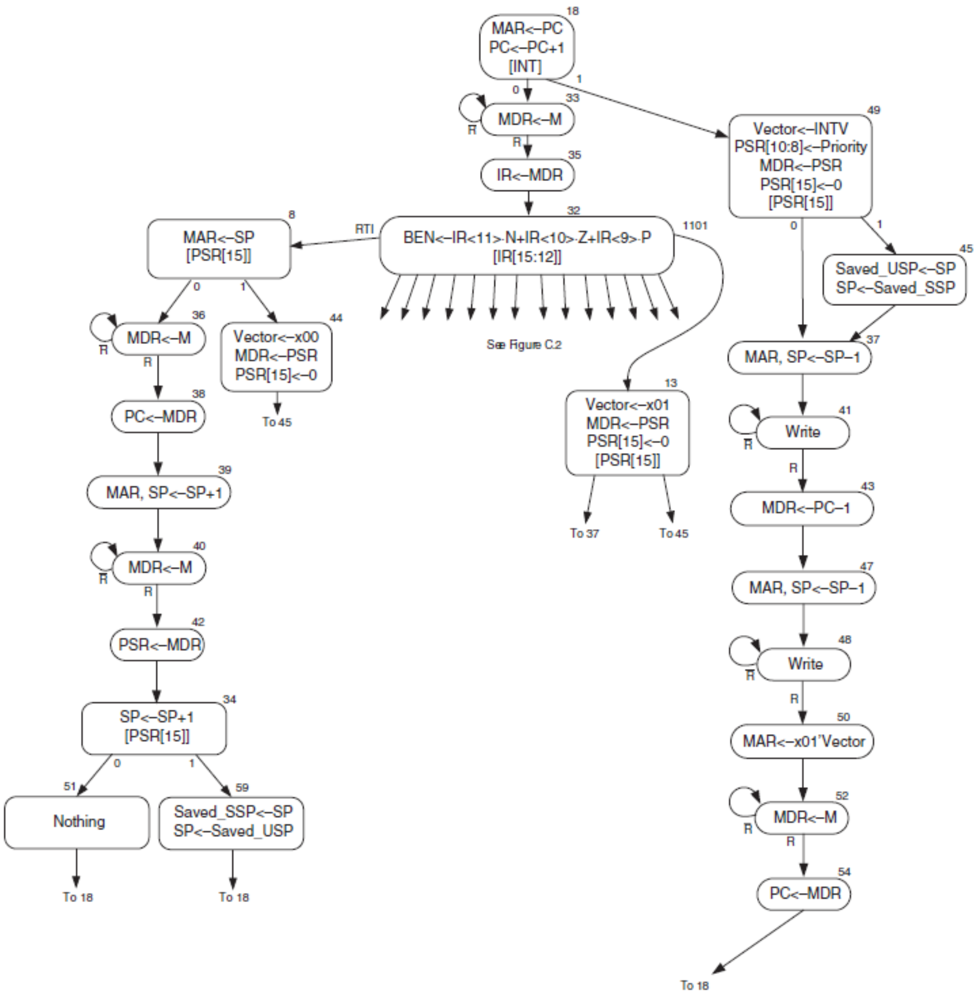
- State 34: Micro-sequencer Control
  - COND=100 ; Test for PSR[15], J[3]
  - J=110011 ; Default next state = 51
  - J=111011 ; Otherwise, next state = 59



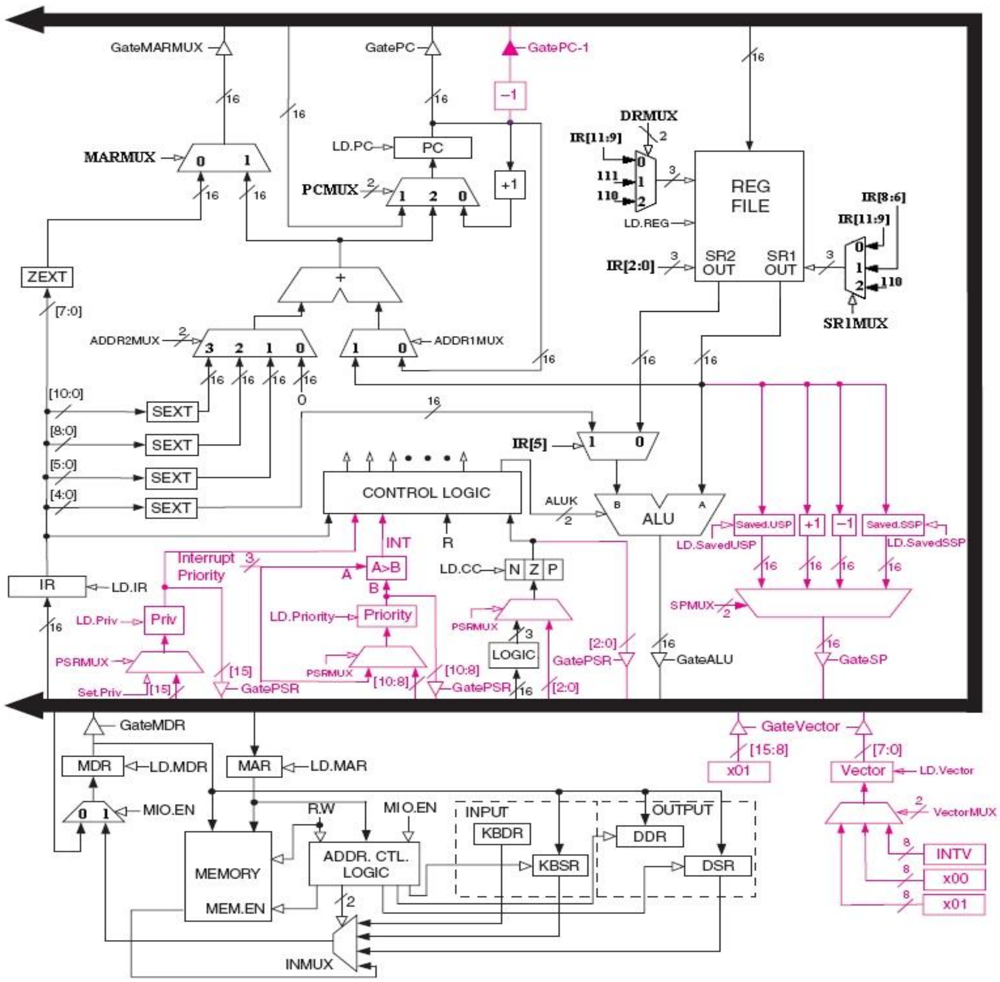
- Processor Status Registry

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

  - Privilege mode (0 – supervisor, 1 – user)
  - Priority level (PL0-PL7)
  - Condition codes (PSR[2] = N, PSR[1]= Z, PSR[0] = P
- LC3 State Machine with Interrupt Support



- LC3 Data Path for Supporting Interrupts
  - Stack registries
    - User stack pointer
    - Supervisory stack pointer
  - POP and PUSH operations
  - Addressing Interrupt Vector Table
  - Privilege and Priority registries
  - NZP register



- Interrupt Handling Control Signals
  - LD.XX
    - LD.SavedUSP, LD.SavedSSP, LD.Vector
    - LD.Priv, (LD.Priority?)
  - XXMUX
    - SPMUX(2), VectorMUX(2)
    - PSRMUX (1)
  - GateXX
    - GateSP, GateVector
    - GatePSR
    - GatePC-1
  - Set.Priv

Control Signal	Number of Signals	Total Number of Bits for Control Signals
GateXX	4	4
LD.XX	4	4
xxMUX	3	5
Set.Priv	1	1
		14 control bits total