Introduction to Embedded Systems

EHB326E

Lectures

Prof. Dr. Müştak E. Yalçın

Istanbul Technical University

mustak.yalcin@itu.edu.tr

Microprocessor interfacing: interrupts

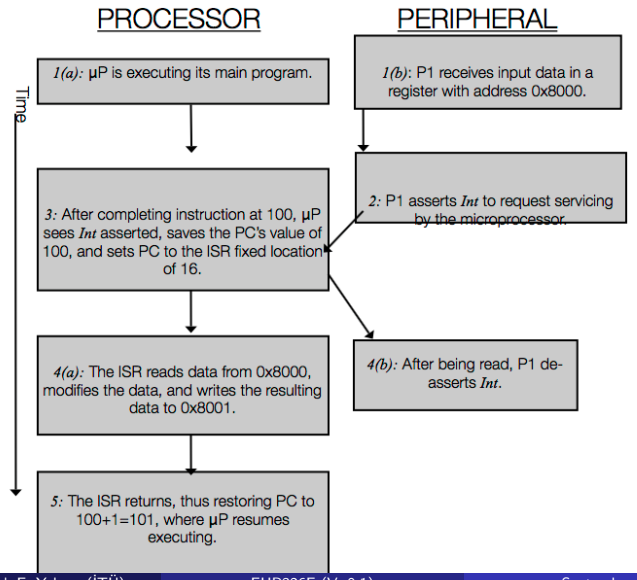
- Suppose a peripheral intermittently receives data, which must be serviced by the processor
  - The processor can *poll* the peripheral regularly to see if data has arrived -wasteful-
  - The peripheral can interrupt the processor when it has data
- Requires an extra pin or pins: `Int`
  - If `Int` is 1, processor suspends current program, jumps to an Interrupt Service Routine, or `ISR`
  - Known as interrupt-driven I/O
  - Essentially, "polling" of the interrupt pin is built-into the hardware, so no extra time!

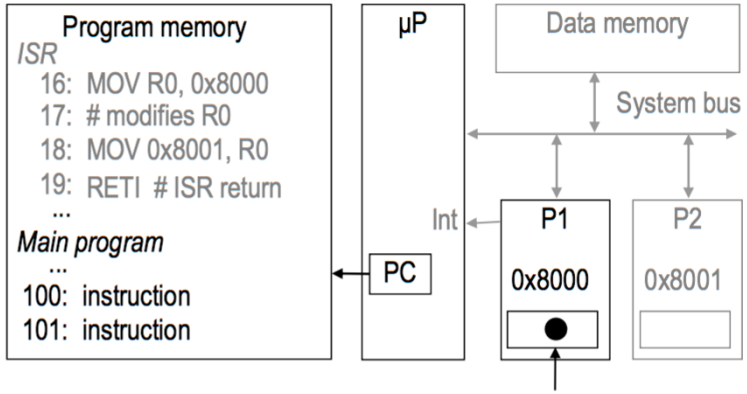
Microprocessor interfacing: interrupts

What is the address (interrupt address vector) of the `ISR`?

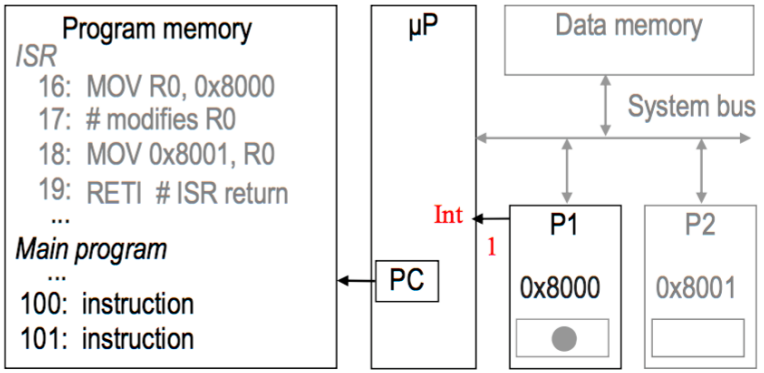
- Fixed interrupt
  - Address built into microprocessor, cannot be changed
  - Either ISR stored at address or a jump to actual `ISR` stored if not enough bytes available
- Vectored interrupt
  - Peripheral must provide the address
  - Common when microprocessor has multiple peripherals connected by a system bus
- Compromise: interrupt address table

Interrupt-Driven I/O Using Fixed ISR Location

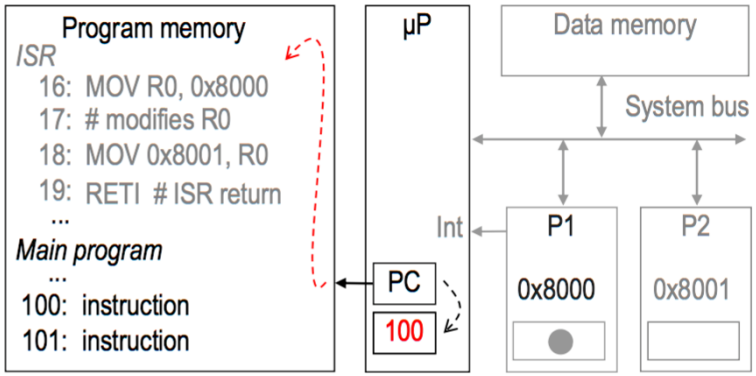




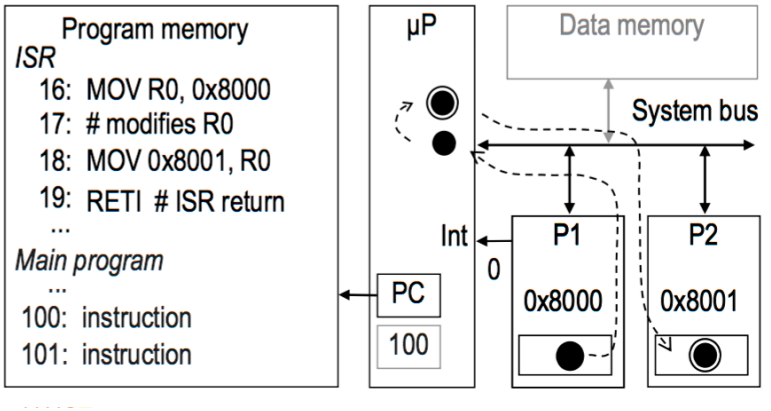
1(a): μP is executing its main program.
1(b): P1 receives input data in a register with address 0x8000h.



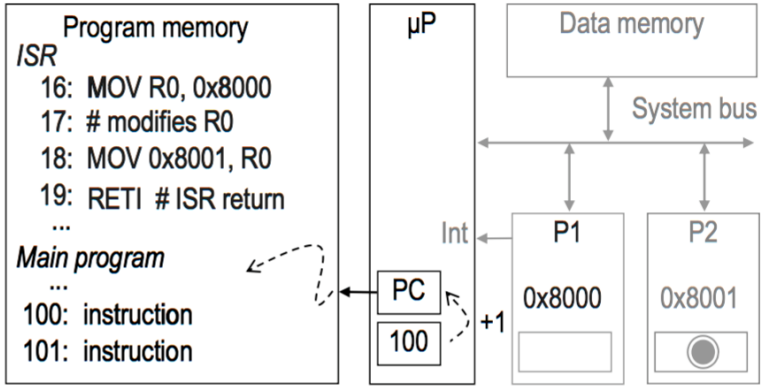
2: P1 asserts Int to request servicing by the microprocessor.



3: After completing instruction at 100h, μP sees Int asserted, saves the PC's value of 100h, and sets PC to the ISR fixed location of 16.

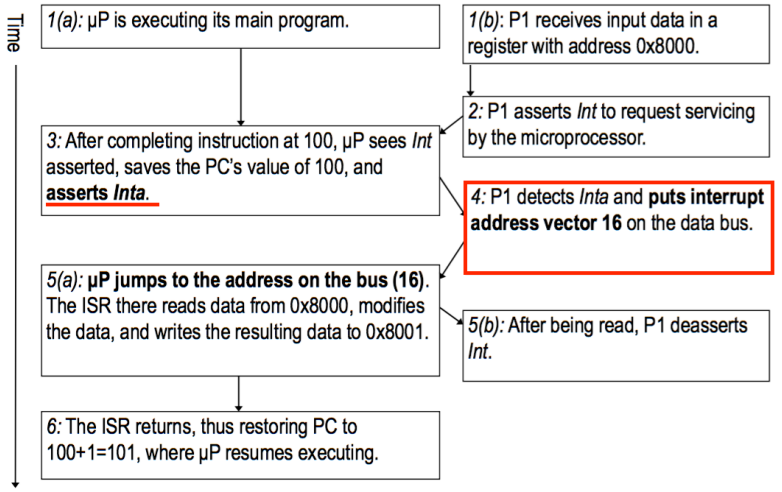


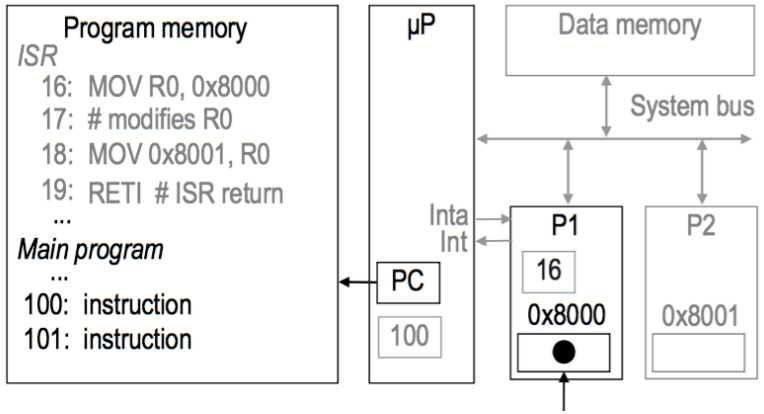
4(a): The ISR reads data from 0x8000h, modifies the data, and writes the resulting data to 0x8001h.
4(b): After being read, P1 deasserts Int.



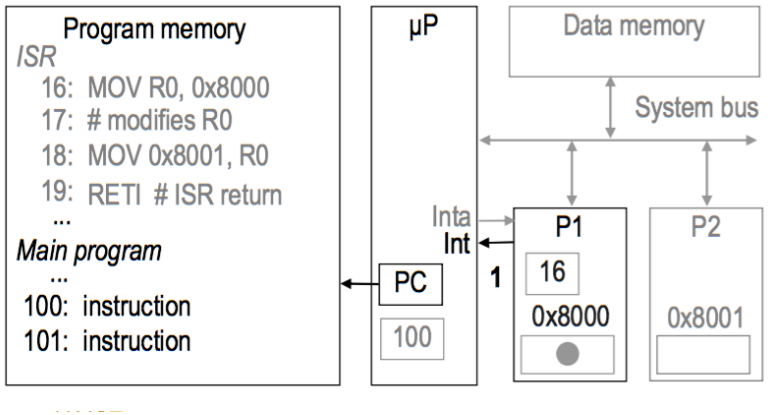
5: The ISR returns, thus restoring PC to $100h + 1 = 101h$*, where* μP *resumes executing.*

Interrupt-Driven I/O Using Vectored Interrupt

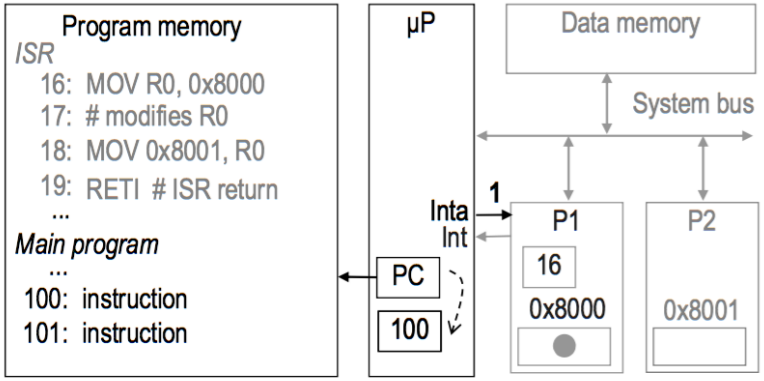




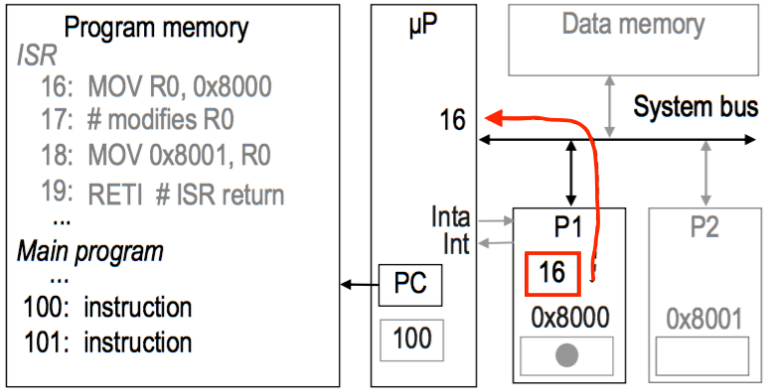
1(a): μP is executing its main program.
1(b): P1 receives input data in a register with address 0x8000h.



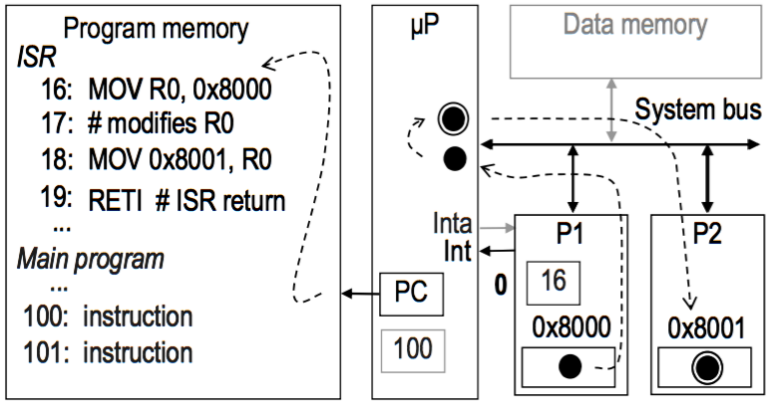
2: P1 asserts Int to request servicing by the microprocessor.



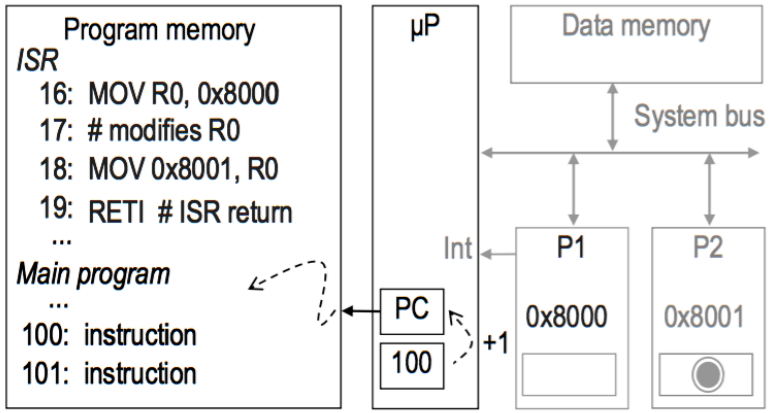
3: After completing instruction at 100*h*, *μP sees Int asserted, saves the PC's value of* 100*h*, *and asserts Inta*



4: P1 detects Inta and puts interrupt address vector 016*h on the data bus*



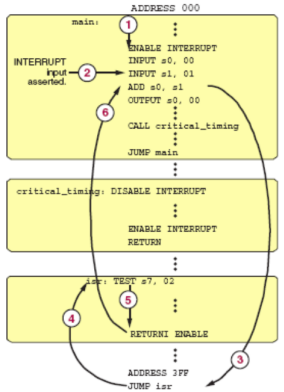
5(a): PC jumps to the address on the bus (16). The ISR there reads data from 0*x*8000*h*, *modifies the data, and writes the resulting data to* 0*x*8001*h*.
5(b): After being read, P1 deasserts Int.



6: The ISR returns, thus restoring the PC to 100*h* + 001=101*h*, *where the μP resumes*

- Compromise between fixed and vectored interrupts
  - One interrupt pin
  - Table (Interrupt address table) in memory holding ISR addresses
  - Peripheral doesn't provide ISR address, but rather index into table
- Maskable vs. non-maskable interrupts
  - Maskable: programmer can set bit that causes processor to ignore interrupt
    - Important when in the middle of time-critical code
  - Non-maskable: a separate interrupt pin that can't be masked
    - Typically reserved for drastic situations, like power failure requiring immediate backup of data to non-volatile memory
- Jump to ISR
  - Some microprocessors treat jump same as call of any subroutine
    - Complete state saved (PC, registers) - may take hundreds of cycles
  - Others only save partial state, like PC only
    - Thus, ISR must not modify registers, or else must save them first
    - Assembly-language programmer must be aware of which registers stored

PicoBlaze



- Branch instructions: Interrupt related instructions

```
eint; // enable interrupt
dint; // disable interrupt
reti disable; // return the interrupt and disable
// interrupt,pc ← STACK[tos]; tos=tos-1;
// c, z← preserved c, z
```

!!!Interrupt signal must be applied at least 2 clock cycles !!

Interrupts of 80C51 ‣ 8051

1. External Interrupt ($\overline{INT0}$,P3.2);
2. Timer 0 Overflow Interrupt
3. External Interrupt ($\overline{INT1}$,P3.3)
4. Timer 1 Overflow Interrupt
5. Serial Port Interrupt
6. Timer 2 Overflow Interrupt

- Interrupt Enable Register IE (Look Fig. 10 in 80C51): Each interrupt source can be individually enabled or disabled by setting or clearing a bit in the SFR named IE (Interrupt Enable).
  Exp: Enable External Int 1 and Timer 0 : MOV IE, #86*h*
  Exp: Disable all interrupts: MOV IE, 0*xxxxxxxb*
- Interrupt Priority Register IP (Look Fig. 11 in 8051): Each interrupt source can also be individually programmed to one of four priority levels by setting or clearing bits in the SFR named IP (Interrupt Priority) and IPH.
  Exp: To place Timer 0 and external Int. 1 at high priority and others at low: MOV IP, #06*h*
- Interrupt Priority Higher Register IPH (Look page 17 and Fig. 12 in 8051)

TCON register is used for programming level or edge triggered interrupt. $\overline{INT0}$ is low level triggered if TCON.0 (IT0)=0. If TCON.0 (IT0)=1, $\overline{INT0}$ is H to L edge triggered. ‣ page 85

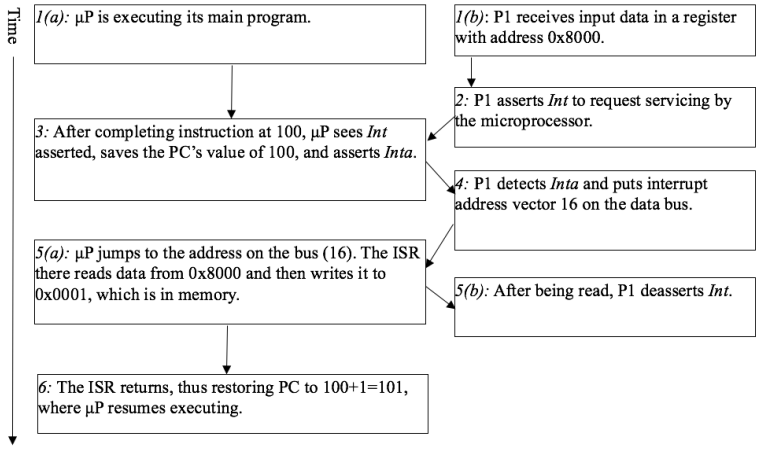
Hardware will set the flag IE0 (TCON.1) and IE1 (TCON.3) for $\overline{INT0}$ and $\overline{INT1}$, respectively. ‣ page 85

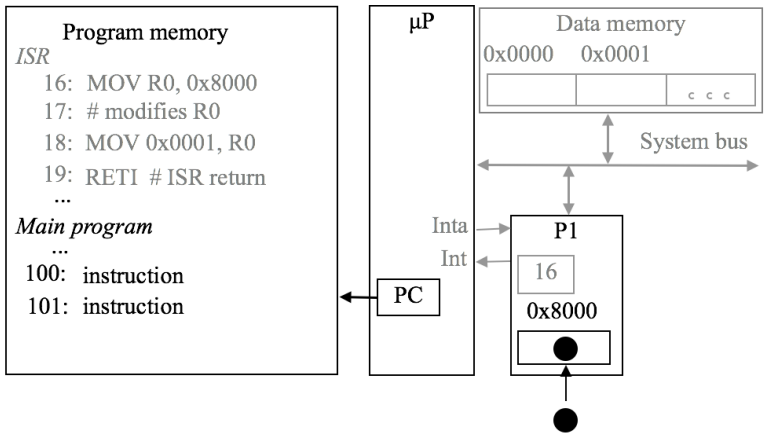
Interrupt adress table: ISR locations 03h, 0Bh, 13h 1Bh, 23h and 2Bh for the sources $\overline{INT0}$, Timer 0, $\overline{INT1}$, Timer 1, Serial and Timer 2, respectively ‣ page 83.

EXP: A sensor send a high to low interrupt on $\overline{INT0}$ then 80C510 asserts P2.0.

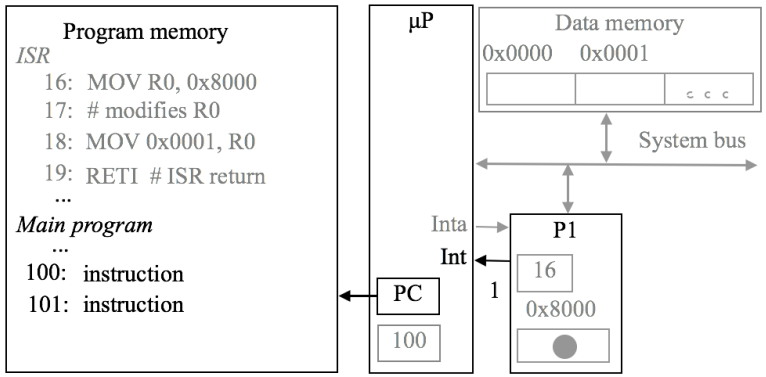
```
                MOV IE, 1000 0001
                MOV IP, 0000 0001
                SJMP INT0_ADRES
030h
INT0_ADRES      SETB P2.0
                RETI
```


Peripheral to memory transfer without DMA, using vectored interrupt

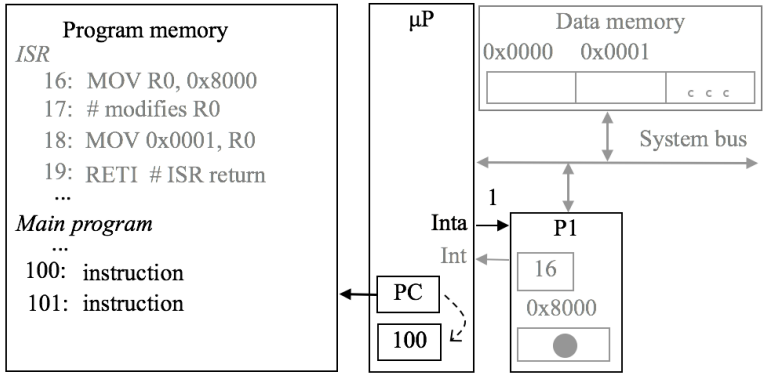




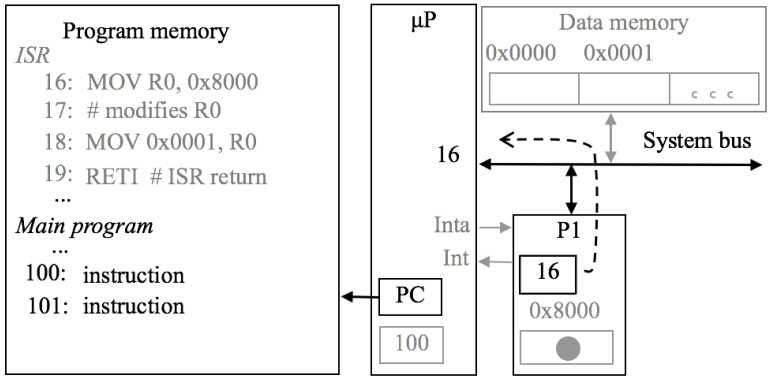
1(a): P is executing its main program
1(b): P1 receives input data in a register with address 0x8000.



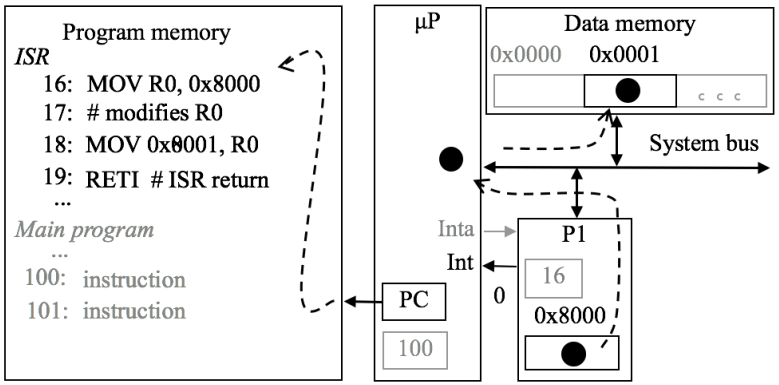
2: P1 asserts Int to request servicing by the microprocessor



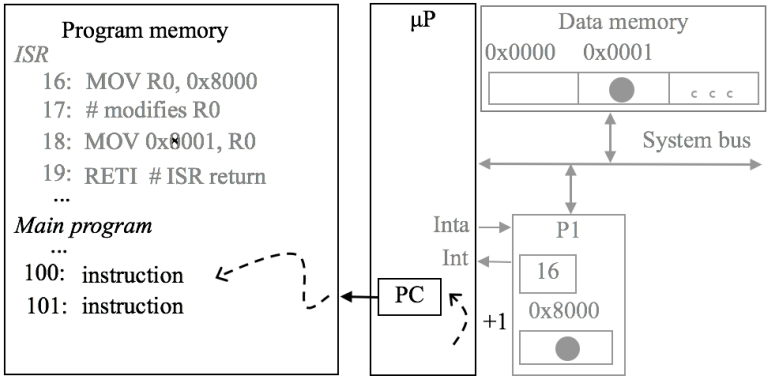
3: After completing instruction at 100, P sees Int asserted, saves the PC's value of 100, and asserts Inta.



4: P1 detects Inta and puts interrupt address vector 16 on the data bus.

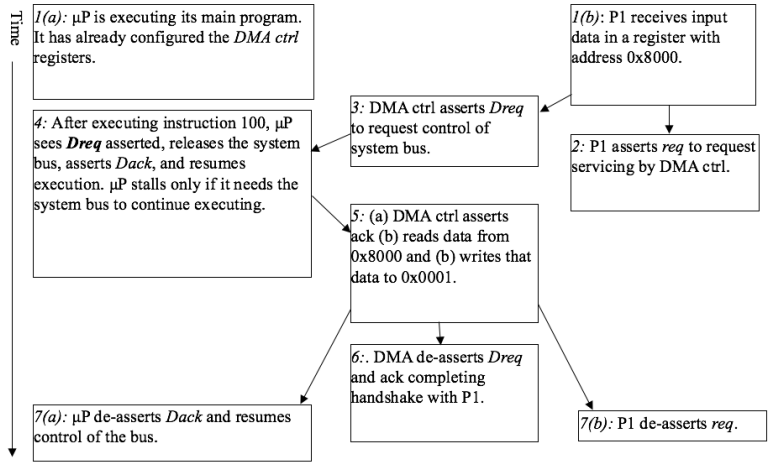


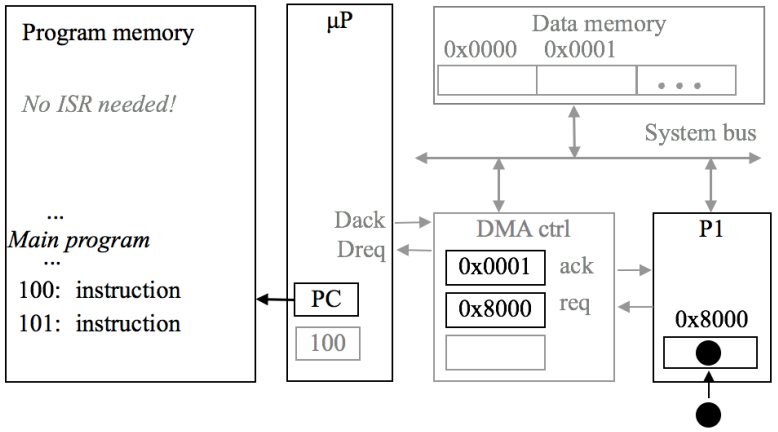
5(a): P jumps to the address on the bus (16). The ISR there reads data from 0x8000 and then writes it to 0x0001, which is in memory.
5(b): After being read, P1 de-asserts Int.



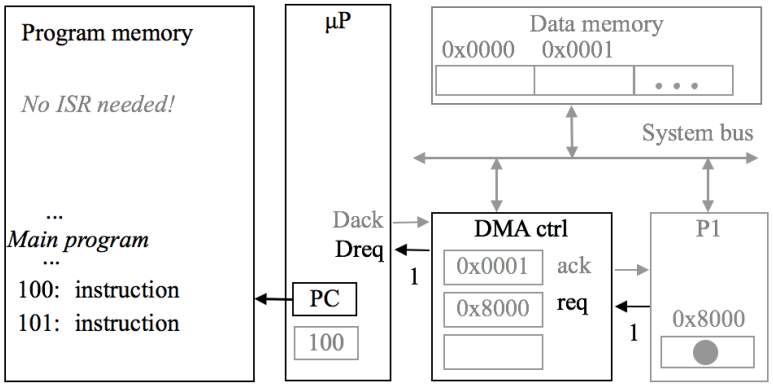
6: The ISR returns, thus restoring PC to 100+1=101, where P resumes executing.

Peripheral to memory transfer with DMA

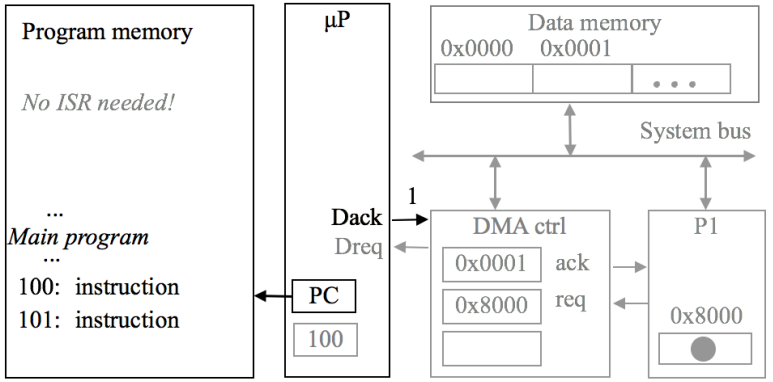




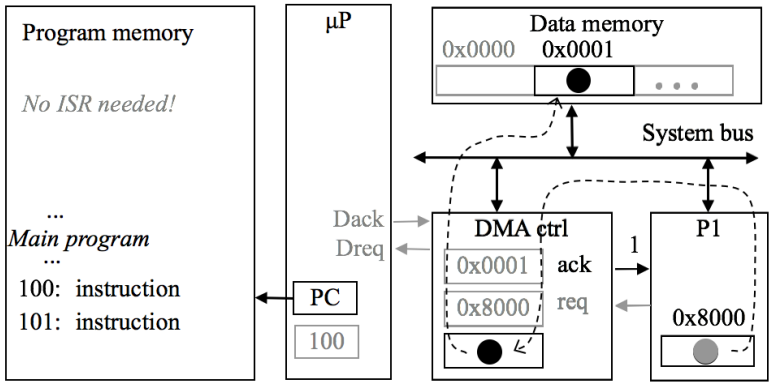
1(a): P is executing its main program. It has already configured the DMA ctrl registers
1(b): P1 receives input data in a register with address 0x8000.



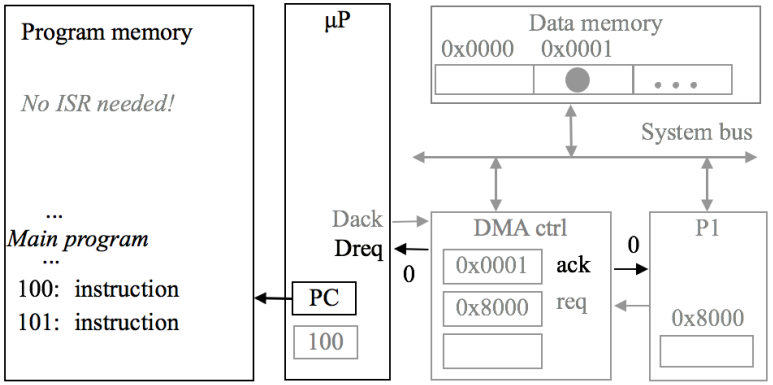
2: P1 asserts req to request servicing by DMA ctrl.
3: DMA ctrl asserts Dreq to request control of system bus



4: After executing instruction 100, P sees Dreq asserted, releases the system bus, asserts Dack, and resumes execution, P stalls only if it needs the system bus to continue executing.



5: DMA ctrl (a) asserts ack, (b) reads data from 0x8000, and (c) writes that data to 0x0001.
(Meanwhile, processor still executing if not stalled!)



6: DMA de-asserts Dreq and ack completing the handshake with P1.