The I/O subsystem

Dr. Ron Shmueli

Chapter 8 Overview

- The I/O subsystem
  - I/O buses and addresses
- Programmed I/O
  - I/O operations initiated by program instructions
- I/O interrupts
  - Requests to processor for service from an I/O device
- Direct Memory Access (DMA)
  - Moving data in and out without processor intervention
- I/O data format change and error control
  - Error detection and correction coding of I/O data

Three Requirements of I/O Data Transmission

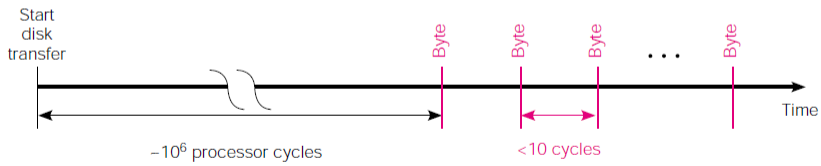
- 1) Data location
  - Correct device must be selected
  - Data must be addressed within that device
- 2) Data transfer
  - Amount of data varies with device & may need be specified
  - Transmission rate varies greatly with device
  - Data may be output, input, or either with a given device
- 3) Synchronization
  - For an output device, data must be sent only when the device is ready to receive it
  - For an input device, the processor can read data only when it is available from the device

Location of I/O Data

- Data location may be trivial once the device is determined
  - Character from a keyboard
  - Character out to a serial printer
- Location may involve searching
  - Record number on a tape drive
  - Track seek and rotation to sector on a disk
- Location may not be simple binary number
  - Drive, platter, track, sector, word on a disk cluster

Fig 8.1 Amount and Speed of Data Transfer

- Keyboard delivers one character about every 1/10 second at the fastest
- Rate may also vary, as in disk rotation delay followed by block transfer



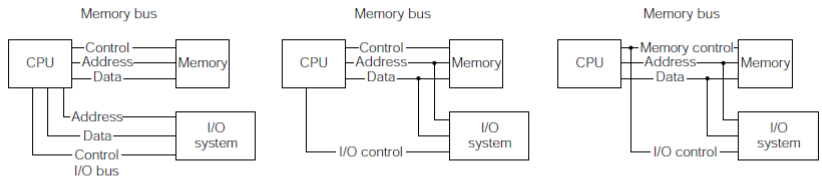
Synchronization—I/O Devices are not Timed by Master Clock

- Not only can I/O rates differ greatly from processor speed, but I/O is asynchronous
- Processor will interrogate state of device and transfer information at clock ticks
- I/O status and information must be stable at the clock tick when it is accessed
- Processor must know when output device can accept new data
- Processor must know when input device is ready to supply new data

Reducing Location and Synch. to Data Transfer

- Since the structure of device data location is device dependent, device should interpret it
  - The device must be selected by the processor, but
  - Location within the device is just information passed to the device
- Synchronization can be done by the processor reading device status bits
  - Data available signal from input device
  - Ready to accept output data from output device
- Speed requirements will require us to use other forms of synchronization: discussed later
  - Interrupts and DMA are examples

Fig 8.2 Separate Memory and I/O Connections to Processor



(a) Separate memory and I/O buses (isolated I/O)

(b) Shared address and data lines

(c) Shared address, data, and control lines (memory-mapped I/O)

- Allows tailoring bus to its purpose, but
- Requires many connections to CPU (pins)

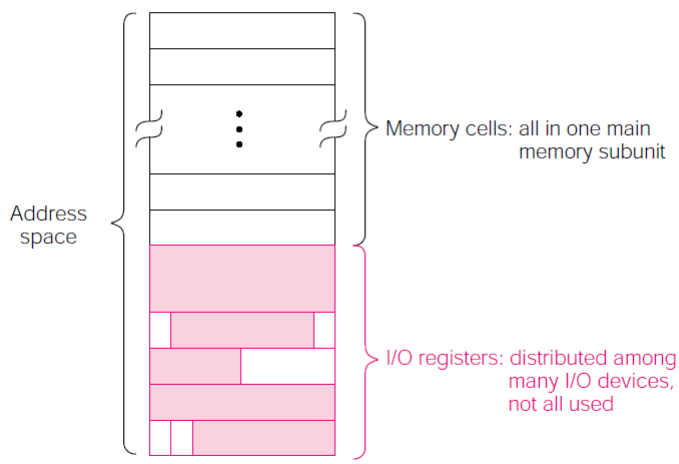
- Memory & I/O access can be distinguished
- Timing and synch. can be different for each

- Least expensive option
- Speed penalty

Memory Mapped I/O

- Combine memory control and I/O control lines to make one unified bus for memory and I/O
- This makes addresses of I/O device registers appear to the processor as memory addresses
- Reduces the number of connections to the processor chip
  - Increased generality may require a few more control signals
- Standardizes data transfer to and from the processor
  - Asynchronous operation is optional with memory, but demanded by I/O devices

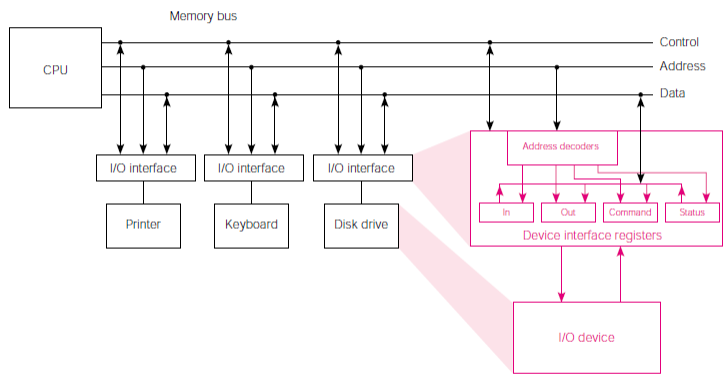
Fig 8.3 Address Space of a Computer Using Memory Mapped I/O



Programmed I/O

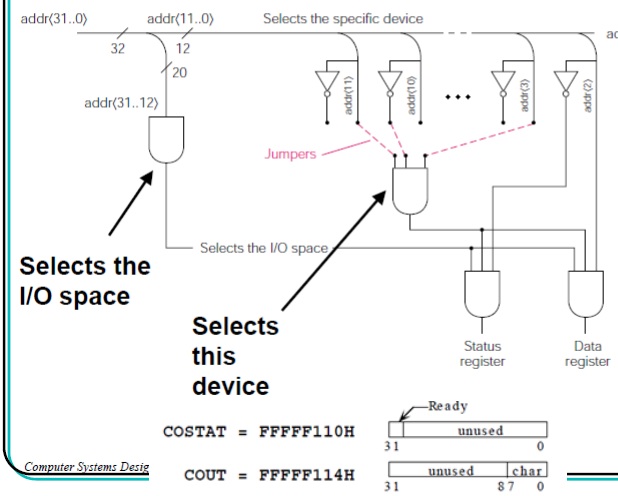
- Requirements for a device using programmed I/O
  - Device operations take many instruction times
  - One word data transfers—no burst data transmission
- Program instructions have time to test device status bits, write control bits, and read or write data at the required device speed
- Example status bits:
  - Input data ready
  - Output device busy or off-line
- Example control bits:
  - Reset device
  - Start read or start write

Fig 8.4 Programmed I/O Device Interface Structure



- Focus on the interface between the unified I/O and memory bus and an arbitrary device Several device registers (memory addresses) share address decode and control logic

Fig 8.5 SRC I/O Register Address Decoder



- Assumes SRC addresses above $FFFFF000_{16}$ are reserved for I/O registers
  - Allows for 1024 registers of 32 bits
  - Is in range $FFFF0000_{16}$ to $FFFFFFFF_{16}$ addressable by negative displacement

הגדרות המימשק לכתיבת חו למדפסת

Example: Programmed I/O Device Driver for Character Output

- Device requirements:
  - 8 data lines set to bits of an ASCII character
  - Start signal to begin operation
  - Data bits held until device returns Done signal
- Design decisions matching bus to device
  - Use low order 8 bits of word for character
  - Make loading of character register signal Start
  - Clear Ready status bit on Start & set it on Done
  - Return Ready as sign of status register for easy testing

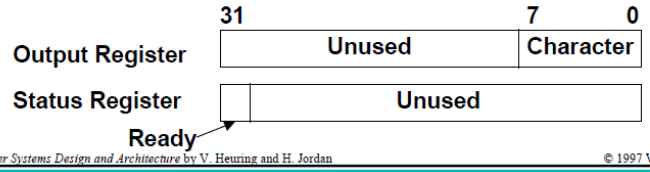
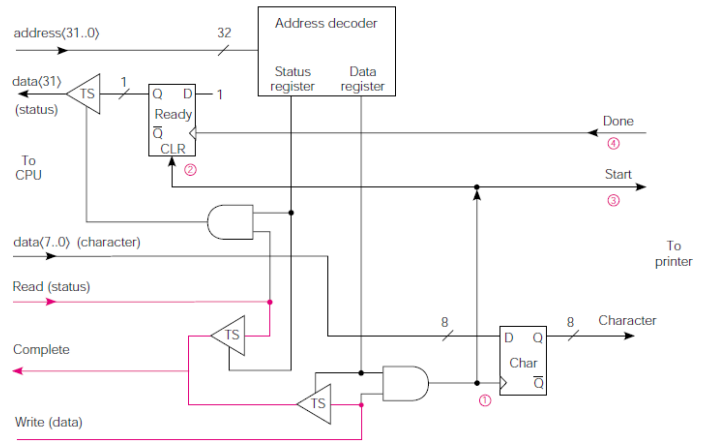


Fig 8.6 Interface Design for SRC Character Output



דרישות הממשק
נתונות בעמוד 8-16

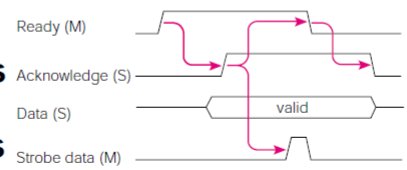
(a) Synchronous input

- Used for register to register inside CPU

(b) Semisynchronous input

- Used for memory to CPU read with few cycle memory

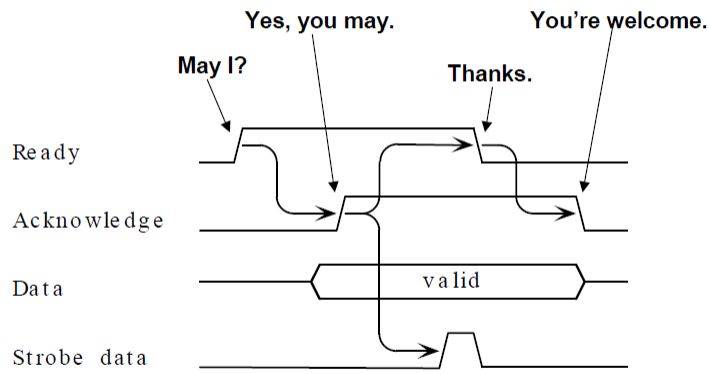
Fig 8.7 Synchronous and Semi-synchronous Data Input



(c) Asynchronous input

- Used for I/O over longer distances (feet) - more-

Fig 8.7c Asynchronous Data input



(c) Asynchronous input

Example: Programmed I/O Device Driver for Character Output

- Device requirements:
  - 8 data lines set to bits of an ASCII character
  - Start signal to begin operation
  - Data bits held until device returns Done signal
- Design decisions matching bus to device
  - Use low order 8 bits of word for character
  - Make loading of character register signal Start
  - Clear Ready status bit on Start & set it on Done
  - Return Ready as sign of status register for easy testing

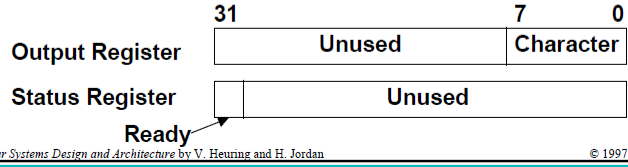
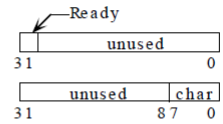


Fig 8.8 Character Output Program Fragment

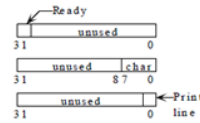
```
Status register   COSTAT = FFFFF110H

Output register    COUT = FFFFF114H

        lar    r3, Wait      ;Set branch target for wait.
        ldr    r2, Char      ;Get character for output.
Wait:   ld     r1, COSTAT    ;Read device status register,
        brpl   r3, r1        ; test for ready, and repeat if not.
        st     r2, COUT      ;Output character and start device.
```

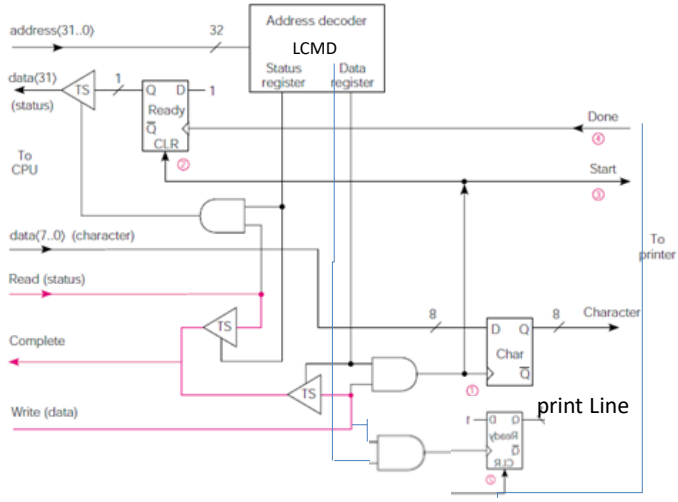


- For readability: I/O registers are all caps., program locations have initial cap., and instruction mnemonics are lower case
- A 10 MIPS SRC would execute 10,000 instructions waiting for a 1,000 character/sec printer

Program Fragment for 80 Character per Line Printer

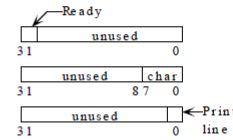
```
Status Register   LSTAT = FFFFF130H
Output Register    LOUT = FFFFF134H
Command Register   LCMD = FFFFF138H
```





Program Fragment for 80 Character per Line Printer

```
Status Register   LSTAT = FFFFF130H
Output Register    LOUT = FFFFF134H
Command Register   LCMD = FFFFF138H
```




```
        lar    r1, Buff      ;Set pointer to character buffer.
        la     r2, 80        ;Initialize character counter and
        lar    r3, Wait      ; branch target.
Wait:   ld     r0, LSTAT     ;Read Ready bit,
        brpl   r3, r0        ; test, and repeat if not ready.
        ld     r0, 0(r1)     ;Get next character from buffer,
        st     r0, LOUT      ; and send to printer.
        addi   r1, r1, 4     ;Advance character pointer, and
        addi   r2, r2, -1    ; count character.
        brnz   r3, r2        ;If not last, go wait for ready.
        la     r0, 1         ;Get a print line command,
        st     r0, LCMD      ; and send it to the printer.
```


Multiple Input Device Driver Software

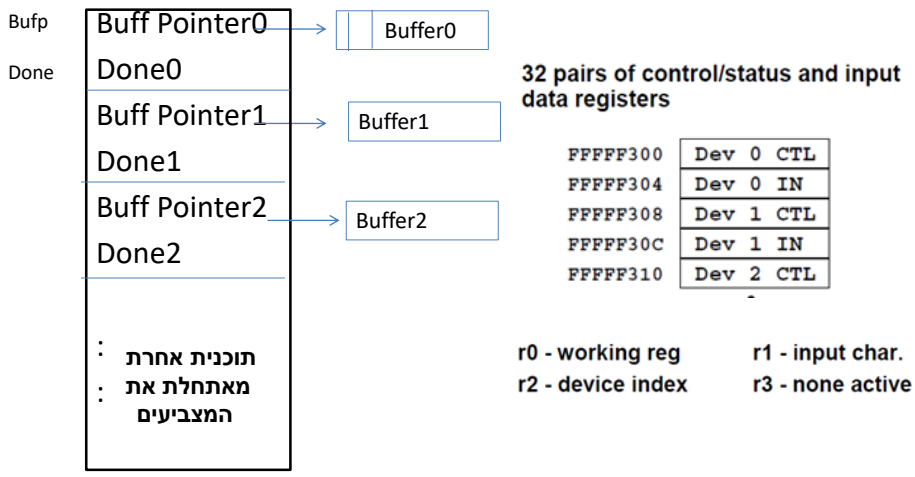
- 32 low speed input devices
  - Say, keyboards at -10 characters/sec
  - Max rate of one every 3 ms
- Each device has a control/status register
  - Only Ready status bit, bit 31, is used
  - Driver works by polling (repeatedly testing) Ready bits
- Each device has an 8 bit input data register
  - Bits 7..0 of 32 bit input word hold the character
- Software controlled by pointer and Done flag
  - Pointer to next available location in input buffer
  - Device's done is set when CR received from device
  - Device is idle until other program (not shown) clears done

דוגמא Multiple Input Device Driver Software

- כתוב מנהל התקן - הקורא תווים מ 32 התקנים איטיים.
- לכל התקן

- מנהל ההתקן יקבל תווים מכל אחד מההתקנים עד לקבלת CR
- המידע שנשלח מכל התקן ישמר בזיכרון במקום מתאים.
- הביצוע ב Pooling

מיבנה הנתונים



bufp מצביע על חוצץ אליו יועברו הנתונים של ההתקן (מאותחל ע"י תוכנית אחרת
DONE מסמן (-1) שההתקן סיים – כלומר התקבל CR מההתקן

Driver Program Using Polling for 32 Input Devices

FFFFF300	Dev 0 CTL
FFFFF304	Dev 0 IN
FFFFF308	Dev 1 CTL
FFFFF30C	Dev 1 IN
FFFFF310	Dev 2 CTL

⋮

- 32 pairs of control/status and input data registers

r0 - working reg r1 - input char.

r2 - device index r3 - none active

```
CICTL   .equ    FFFFF300H       ;First input control register.
CIN     .equ    FFFFF304H       ;First input data register.
CR      .equ    13              ;ASCII carriage return.
Bufp:   .dcw    1               ;Loc. for first buffer pointer.
Done:   .dcw    63              ;Done flags and rest of pointers.
Driver: lar     r4, Next        ;Branch targets to advance to next
        lar     r5, Check       ; character, check device active,
        lar     r6, Start       ; and start a new polling pass.
```


Polling Driver for 32 Input Devices—continued

```
Start:  la      r2, 0           ;Point to first device, and
        la      r3, 1           ; set all inactive flag.
Check:  ld      r0,Done(r2)     ;See if device still active, and
        brmi    r4, r0          ; if not, go advance to next device.
        ld      r3, 0           ;Clear the all inactive flag.
        ld      r0,CICTL(r2)    ;Get device ready flag, and
        brpl    r4, r0          ; go advance to next if not ready.
        ld      r0,CIN(r2)      ;Get character and
        ld      r1,Bufp(r2)     ; correct buffer pointer, and
        st      r0, 0(r1)       ; store character in buffer.
        addi    r1,r1,4         ;Advance character pointer,
        st      r1,Bufp(r2)     ; and return it to memory.
        addi    r0,r0,-CR       ;Check for carriage return, and
        brnz    r4, r0          ; if not, go advance to next device.
        la      r0, -1          ;Set done flag to -1 on
        st      r0,Done(r2)     ; detecting carriage return.
Next:   addi    r2,r2,8         ;Advance device pointer, and
        addi    r0,r2,-256      ; if not last device,
        brnz    r5, r0          ; go check next one.
        brzr    r6, r3          ;If a device is active, make a new pass.
```

Characteristics of the Polling Device Driver

- If all devices active and always have char. ready,
- Then 32 bytes input in 547 instructions
- This is data rate of 585KB/s in a 10MIPS CPU
- But, if CPU just misses setting of Ready, 538 instructions are executed before testing it again
- This 53.8 µsec delay means that a single device must run at less than 18.6Kchars/s to avoid risk of losing data
- Keyboards are thus slow enough

חישוב קצב התקן מקסימלי (בהנחה שתמיד זמין)

סבב קריאה אחד של התוכנית (קריאת 32 מילים מ 32 ההתקנים)

בכניסה 2 פקודות, ביציאה 1 פקודה , בלולאה המרכזית 17 פקודות (מבוצעות 32 פעמים)

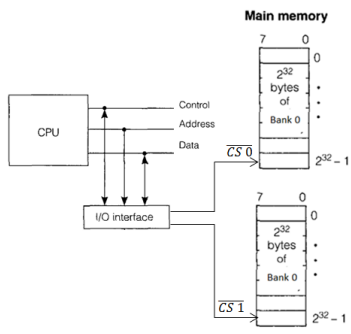
סה"כ 547 =3+ 17x32 פקודות לקריאת 32 מילים

במעבד של 10 MIPS

מס תווים לשנייה $BoudRate = \frac{10MIPSx32}{547}$ =585K$\frac{Byte}{Sec}$

כל התקן שולח 1/32 מהבתים לכן קצב ההתקן לא יעלה על $\frac{585K}{32}$ = 18K $\frac{Byte}{Sec}$

דוגמא: באפליקציה שפותחה סביב מעבד ה SRC בארכיטקטורת 1-BUS התעורר הצורך להכפיל את מרחב הזיכרון. מהנדס המחשבים החליט לתכנן את המערכת ללא ביצוע שינויים במעבד ה SRC (כלומר ארכיטקטורת המעבד וסט הפקודות ללא שינוי). בחירת רכיב הזיכרון הפעיל נעשתה בעזרת מנגנון I/O כמתואר באיור.



ידוע שקיימות מגבלות מסוגים שונים בביצוע של תוכנית.

ביצוע אסמבלר לתוכנית וטעינה לזיכרון, טופלה במקום אחר שאינו חלק מהשאלה

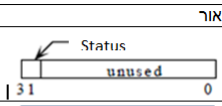
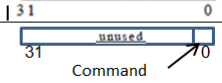
- נתונים נוספים:

רכיבי הזיכרון: שני רכיבי זיכרון מסוג RAM בעלי מרחב זיכרון של 2^{32}x8. (בנק זיכרון 0 ובנק זיכרון 1).

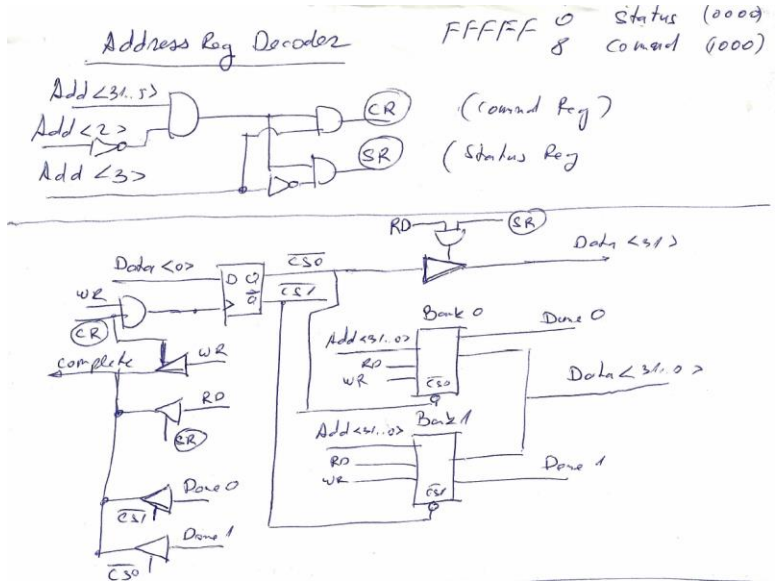
לזיכרון כניסת Chip Select הפעילה ב "0" לוגי.

מיתוג הזיכרונות בעזרת מנגנון קלט פלט כמתואר באיור.

מיפוי קלט פלט:

תיאור	תפקיד	כתובת	שם סימבולי	אופן פעולה
	Status Register	FFFFFFF0	Stat	מסמן את בנק הזיכרון הפעיל 0 = בנק 0 פעיל 1= בנק 1 פעיל
	Command Register	FFFFFFF8	Cmd	הפעלת בנק זיכרון 0 = יופעל בנק 0 1 = יופעל בנק 1

- **תכנן <u>מינימלית</u> את חומרת ה I/O Interface וכולל Address Decoder. ואת חיבור רכיבי הזיכרון ל CPUבהתאם לאיור המופיע למעלה. הראה וסמן בברור את כל הקווים**



(5 נק) בהתייחס לעובדות הבאות:

לאחר ביצוע פעולת Reset למעבד ה SRC, ה PCמקבל את הכתובת PC=00000000H.

התוכנית שנכתבה עבור המעבד מורחב הזיכרון- מתחילה בכתובת 100H של בנק זיכרון 0.

לא תוכנן מנגנון RESETבחומרה לבחירת בנק הזיכרון ההתחלתי (לאחר Reset לא ברור איזה בנק זיכרון פעיל)

כתוב את קטעי התוכנית הנדרשים בשפת אסמבלי של ה SRC, כל שלאחר Reset התוכנית תתחיל מכתובת 100 של בנק זיכרון 0. ציין במפורש כתובות אבסולוטיות של תוכניתך.

Bank #	Address	Labels	Commands	Comments
		Stat	.equ FFFFFFF0	כתובת Status
		CMD	.equ FFFFF8	כתובת Command
0	0		lar R2, Start	
	4		br R2	
	8		lar R2, Start	
	12		br R2	
	⋮			
0	100	start:		
1	0		la R1, 0	
	4		St R1, CMD	

(5 נק) במהלך התוכנית, לאחר סיום הפקודה המתבצעת בכתובת 000002000H (כתובת בבנק 0).
נדרש לבצע הסתעפות לתווית NEXTהנמצאת כתובת 100003000H (כתובת בבנק 1), כתוב את קטע התוכנית לביצוע ההסתעפות (התייחס לכל הפקודות הנדרשות – בכל הכתובות המתאימות).

	Bank #	Address	Labels	Commands	Comments
			Start	equ F...FFF0	
			CMD	equ F FFF8	
Bank 0	0	2004		la R1, 1	
	0	2008		St R1 CMD	
Bank 1	1	200C		lar R2, Next	
		2010		br R2	
		⋮			
		⋮			
	1	3000	Next:		

במהלך התוכנית, לאחר סיום הפקודה המתבצעת בכתובת 100003000H (כתובת בבנק 1).
נדרש לקרא נתון הנמצא בכתובת 000001000H (כתובת בבנק 0), ולהמשיך בביצוע התוכנית כתוב את קטע התוכנית (התייחס לכל הפקודות הנדרשות – בכל הכתובות המתאימות).
הראה באיזה כתובת תמשיך התוכנית המתבצעת בבנק 1 לאחר קריאת הנתון

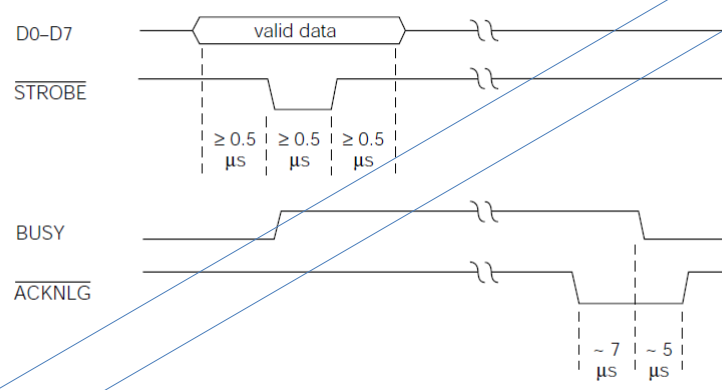
Bank #	Address	Labels	Commands	Comments
		Start	equ F---F0	Start אזור
		CMD	equ F---F8	CMD אזור
0	100	Dat	dw 1000	
1	3004		la R1, 0	
1	3008		st R1, CMD	
1	300C		NOP	
1	3010		NOP	
1	3014		NOP	
1	3018			כאן אמשיך כיצוא המידע

0	300C		lar R2, Data	
0	3010		la R1, A	
0	3014		st R1 CMD	

Tbl 8.1 The Centronics Printer Interface

Name	In/Out	Description
$\overline{\text{STROBE}}$	Out	Data out strobe
D0	Out	Least significant data bit
D1	Out	Data bit
...	...	...
D7	Out	Most significant data bit
$\overline{\text{ACKNLG}}$	In	Pulse on done with last char.
BUSY	In	Not ready
$\overline{\text{PE}}$	In	No paper when high
SLCT	In	Pulled high
$\overline{\text{AUTOFEEDXT}}$	Out	Auto line feed
$\overline{\text{INIT}}$	Out	Initialize printer
$\overline{\text{ERROR}}$	In	Can't print when low
$\overline{\text{SLCTIN}}$	Out	Deselect protocol

Fig 8.11 Centronics Interface Timing



- Minimum times specified for output signals
- Nominal times specified for input signals

SRC Exceptions -חריגים

- Exception - הפרעה לזרימת התוכנית
- בספר שלנו - reset +Interrupts

2-42 Chapter 2 - Machines, Machine Languages, and Digital Logic

Using RTN to describe the SRC (static) Processor State

Processor state

PC⟨31..0⟩:	program counter (memory addr. of next inst.)
IR⟨31..0⟩:	instruction register
Run:	one bit run/halt indicator
Strt:	start signal
R[0..31]⟨31..0⟩:	general purpose registers

- תזכורת
- Run אות פנימי ל SRC.
- Strt - חיצוני ל SRC

2-53 Chapter 2 - Machines, Machine Languages, and Digital Logic

SRC RTN—The Main Loop

ii := instruction_interpretation:
ie := instruction_execution :

```
ii := ( ¬Run∧Strt → Run ← 1:
        Run → (IR ← M[PC]: PC ← PC + 4;
        ie) );
ie := (
    ld (:= op= 1) → R[ra] ← M[disp]:
    ldr (:= op= 2) → R[ra] ← M[rel]:
    . . .
    stop (:= op= 31) → Run ← 0:
);  ii
```

Big switch statement on the opcode

Thus ii and ie invoke each other, as coroutines.

Computer Systems Design and Architecture by V. Heuring and H. © 1997 V. Heuring and H. Jordan, Updated January, 2001 by David M. Zar

Abstract RTN for SRC Reset and Start

Processor State

Strt: Start signal
Rst: External reset signal

instruction_interpretation := (
¬Run∧Strt → (Run ← 1: PC, R[0..31] ← 0); Hard reset
Run∧¬Rst → (IR ← M[PC]: PC ← PC + 4;
instruction_execution): Normal fetch
Run∧Rst → (Rst ← 0: PC ← 0); instruction_interpretation):
Soft reset

Tbl 4.17 Concrete RTN Describing Reset During add Instruction Execution

Step	Concrete RTN
T0	¬Reset → (MA ← PC: C ← PC + 4):
	Reset → (Reset ← 0: PC ← 0: T ←0):
T1	¬Reset → (MD ← M[MA]: P ← C):
	Reset → (Reset ← 0: PC ← 0: T ← 0):
T2	¬Reset → (IR ← MD):
	Reset → (Reset ← 0: PC ← 0: T ← 0):
T3	¬Reset → (A ← R[rb]):
	Reset → (Reset ← 0: PC ← 0: T ← 0):
T4	¬Reset → (C ← A + R[rc]):
	Reset → (Reset ← 0: PC ← 0: T ← 0):
T5	¬Reset → (R[ra] ← C):
	Reset → (Reset ← 0: PC ← 0: T ← 0):

I/O Interrupts

- Key idea: instead of processor executing wait loop, device requests interrupt when ready
- In SRC the interrupting device must return the vector address and interrupt information bits
- Processor must tell device when to send this information—done by acknowledge signal
- Request and acknowledge form a communication handshake pair
- It should be possible to disable interrupts from individual devices

עקרונות בסיסיים לטיפול בחריגים (פסיקות)

- ההתקן שולח $\overline{\text{ireq}}$ (לוגיקה שלילית open collector)
- עם קבלת הפסיקה
  - המעבד ישלים את ביצוע הפקודה הנוכחית- ויענה ב iack
  - ישמור כתובת חזרה +נטרול אפשרות לחריגים.
  - ביצוע הסתעפות ל- ISR (Interrupt Service Routine) (כתובת ה ISR תועבר ע"י ההתקן)
- תפקידי שגרת הפסיקה ISR
  - בכניסה ל ISR
    - שמירת תוכן האוגרים (+מילת מצב – אין ב SRC)
    - אפשור קבלת חריגים (קינון חריגים) , בדר"כ חריגים בעלי עדיפות גבוהה יותר
  - מתן שרות להתקן
  - ביציאה מה ISR
    - נטרול קבלת חריגים
    - שחזור אוגרים (+מילת מצב)
    - חזרה לתוכנית (RETI/rfi- תאפשר פסיקות נוספות)

General Comments on Exceptions

- An exception is an event that causes a change in the program specified flow of control
- Because normal program execution is interrupted, they are often called interrupts
- We will use exception for the general term and use interrupt for an exception caused by an external event, such as an I/O device condition
- The usage is not standard. Other books use these words with other distinctions, or none

Combined Hardware/Software Response to an Exception

- The system must control the type of exceptions it will process at any given time
- The state of the running program is saved when an allowed exception occurs
- Control is transferred to the correct software routine, or "handler" for this exception
- This exception, and others of less or equal importance are disallowed during the handler
- The state of the interrupted program is restored at the end of execution of the handler

תהליך ביצוע פסיקה ב SRC

- התקן שולח
  - Isrc_info + Isec_vec + IREQ
- המעבד מבצע
  - iack
  - IPC← PC
  - II←Isrc_info
  - IE←0
  - PC←Ivec (Ivec←Isrc_vec (shifed left by 4))

מבצע ISR שמסתיימת בפקודה RFI

מבצע RFI

» PC←IPC

» IE←1

ביצוע פסיקה ב SRC

- כאשר התקן צריך שרות (וה IE=1 הפנימי שלו)
  - התקן שולח IREQ.
  - מעביר ב DATA BUS את המידע הבא:
    - ב Data<23..16> תועבר כתובת ה ISR ב vect<7..0>
      ימופה ב SRC ל 0000 vec<7..0> 0000...00 = IVEC
    - ב Data<15..0> יועבר info<15..0> מידע נוסף על ההתקן
- ה SRC מגיב לפסיקה במידה אם IE=1
  - שולח IACK לאישור קבלת הפסיקה (ובזמן הזה קורא את ה DataBus)
  - שומר כתובת חזרה PC ← IPC
  - שומר מידע על ההתקן ISR_info ← II<15..0>
  - לא מאפשר פסיקות נוספות 0→ IE
  - הסתעפות ל ISR ע"י IVEC ←PC
    - ביצוע ה ISR
    - וחזרה לתוכנית בעזרת RFI (IE←1 PC←IPC)

SRC Processor State Associated with Interrupts

Processor interrupt mechanism

From Dev. →	ireq:	interrupt request signal
To Dev. →	iack:	interrupt acknowledge signal
Internal →	IE:	one bit interrupt enable flag
to CPU →	IPC⟨31..0⟩:	storage for PC saved upon interrupt
" →	II⟨31..0⟩:	info. on source of last interrupt
From Dev. →	Isrc_info⟨15..0⟩:	information from interrupt source
From Dev →	Isrc_vect⟨7..0⟩:	type code from interrupt source
Internal →	Ivect⟨31..0⟩:= 20@0#Isrc_vect<7..0>#4@0:	defines an interrupt vector table

Ivect⟨31..0⟩

000 . . . 0	Isrc_vect⟨7..0⟩	0000
31 12	11 4	3 0

SRC Instruction Interpretation Modified for Interrupts

instruction_interpretation :=
(¬Run∧Strt → Run ← 1:
Run∧¬(ireq∧IE) → (I ← M[PC]: PC ← PC + 4; instruction_execution);
Run∧(ireq∧IE) → (IPC ← PC⟨31..0⟩:
II⟨15..0⟩ ← Isrc_info⟨15..0⟩: iack ← 1:
IE ← 0: PC ← Ivect⟨31..0⟩; iack ← 0);
instruction_interpretation);

- If interrupts are enabled, PC and interrupt info. are stored in IPC and II, respectively
  - With multiple requests, external priority circuit (discussed in later chapter) determines which vector & info. are returned
- Interrupts are disabled
- The acknowledge signal is pulsed

complete instruction interpretation definition, including hard and soft reset, exceptions, and normal instruction fetch

```
instruction_interpretation :=
(¬Run∧Strt → (Run ← 1: PC, R[0..31] ← 0;          Hard reset
            instruction_interpretation):
Run∧Rst → ( Rst ← 0: IE ← 0: PC ← 0;              Soft reset
            instruction_interpretation):
Run∧¬Rst∧(ireq∧IE) → (IPC ← PC⟨31..0⟩:            Interrupt
            II⟨15..0⟩ ← Isrc_info⟨15..0⟩:
            IE ← 0: PC ← Ivect⟨31..0⟩:
            iack ← 1; iack ← 0;
            instruction_interpretation):
Run∧¬Rst∧¬(ireq∧IE)→ (IR ← M[PC]:                 Normal fetch
        PC ← PC + 4; instruction_execution):
```


SRC Instructions to Support Interrupts

Return from interrupt instruction

rfi (:= op = 29) → (PC ← IPC: IE ← 1):

Save and restore interrupt state

svi (:= op = 16) → (R[ra]⟨15..0⟩ ← II⟨15..0⟩: R[rb] ← IPC⟨31..0⟩):

ri (:= op = 17) → (II⟨15..0⟩ ← R[ra]⟨15..0⟩ : IPC⟨31..0⟩ ← R[rb]):

Enable and disable interrupt system

een (:= op = 10) → (IE ← 1):

edi (:= op = 11) → (IE ← 0):

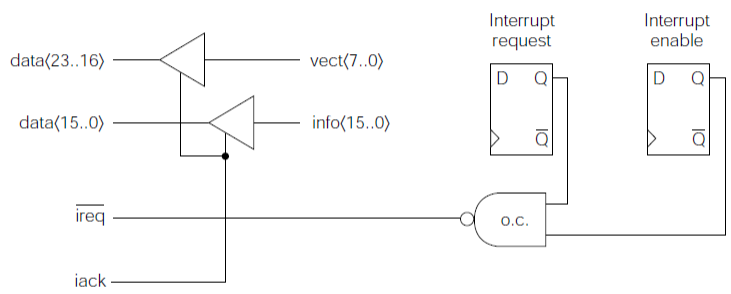
- **The 2 rfi actions are indivisible, can't een & branch**

Concrete RTN for SRC Instruction Fetch with Interrupts

Step	¬(ireq∧IE) Concrete RTN	(ireq∧IE)
T0.	(¬(ireq∧IE) → (MA ← PC: C ← PC+4):	(ireq∧IE) → (IPC ← PC: II ← Isrc_info: IE ← 0: PC← 22@0#Isrc_vect⟨7..0⟩#00: Iack←1; Iack ← 0: End);
T1.	MD ← M[MA] : PC ← C;	
T2.	IR ← MD;	

- PC could be transferred to IPC over the bus
- II and IPC probably have separate inputs for the externally supplied values
- *iack* is pulsed, described as ←1; ←0, which is easier as a control signal than in RTN

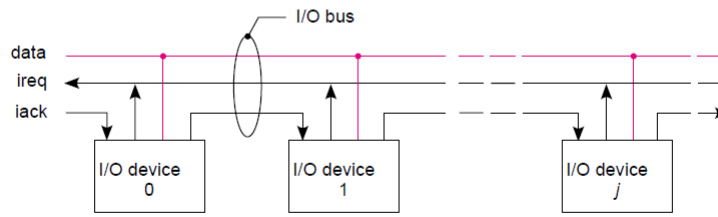
Fig 8.12 Simplified Interrupt Interface Logic



- Request and enable flags per device
- Returns vector and interrupt information on bus when acknowledged

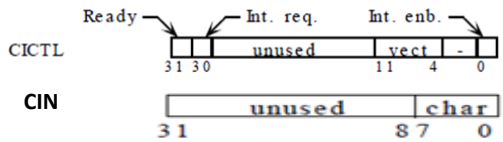
Fig 8.13 Daisy-Chained Interrupt Acknowledge Signal

- How does acknowledge signal select one and only one device to return interrupt info.?
- One way is to use a priority chain with acknowledge passed from device to device



דוגמא-פסיקות

- התקן דמוי מקלדת מחובר ל CPU ושלוח תווים.
  התוכנית תקלוט תווים עד לקבלת <CR>
- האוגרים של ההתקן



- ארגון נתונים ב SRC

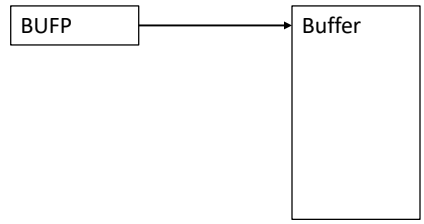
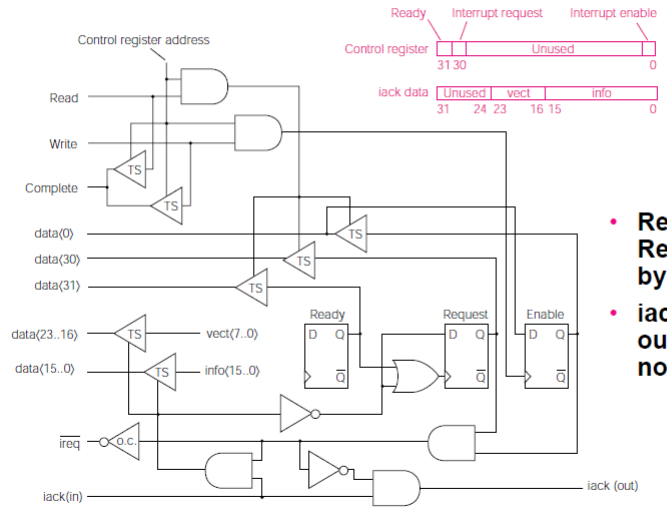


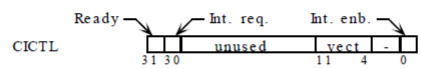
Fig 8.14 Interrupt Logic for an SRC I/O Interface



- Request set by Ready, cleared by acknowledge
- iack only sent out if this device not requesting

Getline Subroutine for Interrupt Driven Character I/O

תוכנית לאתחול ההתקן




```
;Getline is called with return address in R31 and a pointer to a
;character buffer in R1. It will input characters up to a carriage
;return under interrupt control, setting Done to -1 when complete.
CR      .equ    13              ;ASCII code for carriage return.
CIvec   .equ    01F0H           ;Character input interrupt vector address.
Bufp:   .dw     1               ;Pointer to next character location.
Save:   .dw     2               ;Save area for registers on interrupt.
Done:   .dw     1               ;Flag location is -1 if input complete.
Getln:  st      r1, Bufp        ;Record pointer to next character.
        edi                     ;Disable interrupts while changing mask.
        la      r2, 1F1H        ;Get vector address and device enable bit
        st      r2, CICTL       ; and put into control register of device.
        la      r3, 0           ;Clear the
        st      r3, Done        ; line input done flag.
        een                     ;Enable Interrupts
        br      r31             ;   and return to caller.
```

ISR

Interrupt Handler for SRC Character Input

```
        .org    CIvec           ;Start handler at vector address.
        str     r0, Save        ;Save the registers that
        str     r1, Save+4      ; will be used by the interrupt handler.
        ldr     r1, Bufp        ;Get pointer to next character position.
        ld      r0, CIN         ;Get the character and enable next input.
        st      r0, 0(r1)       ;Store character in line buffer.
        addi    r1, r1, 4       ;Advance pointer and
        str     r1, Bufp        ; store for next interrupt.
        lar     r1, Exit        ;Set branch target.
        addi    r0,r0, -CR      ;Carriage return? addi with minus CR.
        brnz    r1, r0          ;Exit if not CR, else complete line.
        la      r0, 0           ;Turn off input device by
        st      r0, CICTL       ; disabling its interrupts.
        la      r0, -1          ;Get a -1 indicator, and
        str     r0, Done        ; report line input complete.
Exit:   ldr     r0, Save        ;Restore registers
        ldr     r1, Save+4      ; of interrupted program.
        rfi                     ;Return to interrupted program.
```


General Functions of an Interrupt Handler

1) Save the state of the interrupted program
2) Do programmed I/O operations to satisfy the interrupt request
3) Restart or turn off the interrupting device
4) Restore the state and return to the interrupted program

Interrupt Response Time

- Response to another interrupt is delayed until interrupts re-enabled by rfi
- Character input handler disables interrupts for a maximum of 17 instructions
- If the CPU clock is 20MHz, it takes 10 cycles to acknowledge an interrupt, and average execution rate is 8 CPI

Then 2nd interrupt could be delayed by

$$(10 + 17 \times 8) / 20 = 7.3\mu sec$$

קלדנית -מקלידה 120 מילים בדקה -בהנחה שכל מילה 5 תווים,
5 * 120/60 = 10 char/sec
7.3*10 /10^6= 7.3*10^(-5) = 0.0073% המעבד יהיה עסוק

בכמה קלדניות יוכל לטפל ? --- 100/0.0073