Microprocessor Applications

Prof M.P.Gough
m.p.gough@sussex.ac.uk

Syllabus :

Microprocessor architecture,
Organisation & operation of microcomputer systems.
Hardware and software interaction.
Programme and data storage.
Parallel interfacing and programmable ICs.
Serial interfacing, standards and protocols.
Analogue interfacing. Interrupts and DMA.
Microcontrollers and small embedded systems.
The CPU, memory and the operating system.

Teaching Methods:

2 lectures / week

Exercises/examples reviewed in workshops
weeks 3,5,7,9

Research for handed in assignment (Week 10)

Assessment:

Written Assignment	20% Week 10 (March)
Unseen Examination	80% June

Reading List

- Alan Clements. 2000. The Principles of Computer Hardware, Oxford, 3rd edition. (A number are available for loan from the Engineering & Design Department Office)

- For assessment exercise: Various manufacturer's microprocessor and microcontroller datasheets and user documentation downloadable from the internet.

- For lecture notes I have used the above Clements + others below

Note: these are not recommended for buying

The 68000 Microprocessor Family, M.A.Miller,1992 MacMillan

Digital Fundamentals, Floyd, 2006, Pearson International

Computer Engineering Hardware Design, M.Manno, Prentice Hall

Microcomputer Interfacing, H.Stone, ddison Wesley

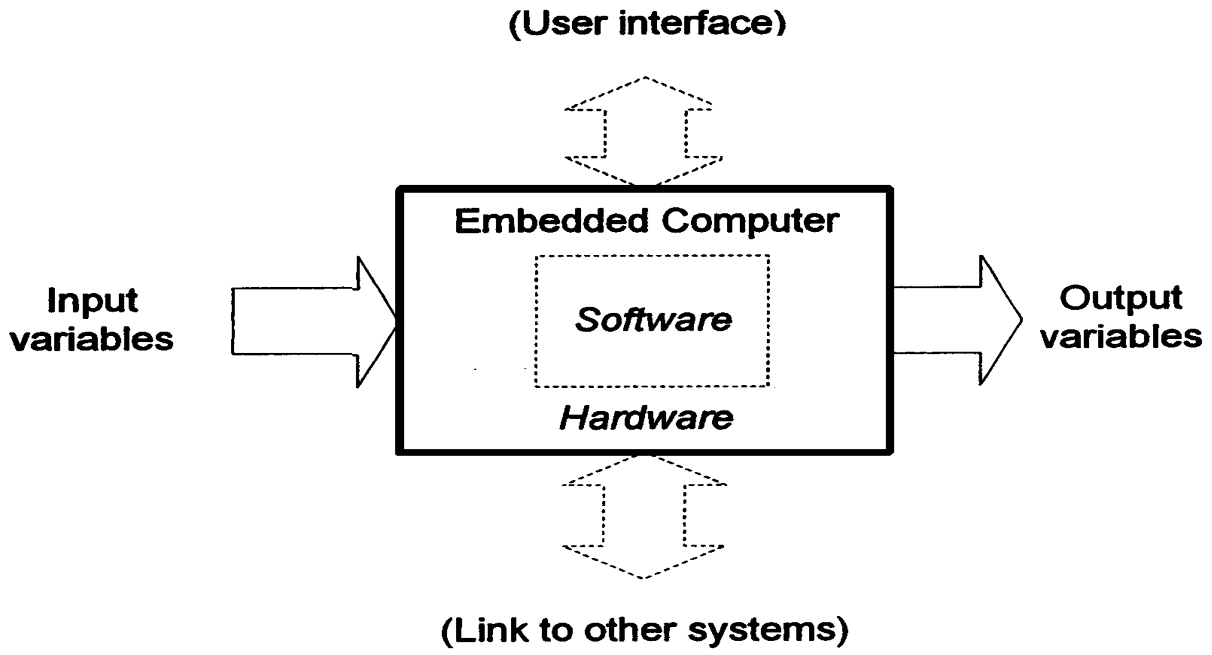
+ various datasheets from web

e.g. 68HC000 =CMOS 68000 version

Course comprised of 8 topics:

1. **Review Architecture & Programming of Microcomputer Systems**
2. **Programme and Data Storage**
3. **Parallel Input & Output Peripheral Devices**
4. **Interrupts**
5. **Serial Input and Output**
6. **Analogue I/O**
7. **Microcontrollers for Small Embedded Systems**
8. **CPU, Memory, and the Operating System**

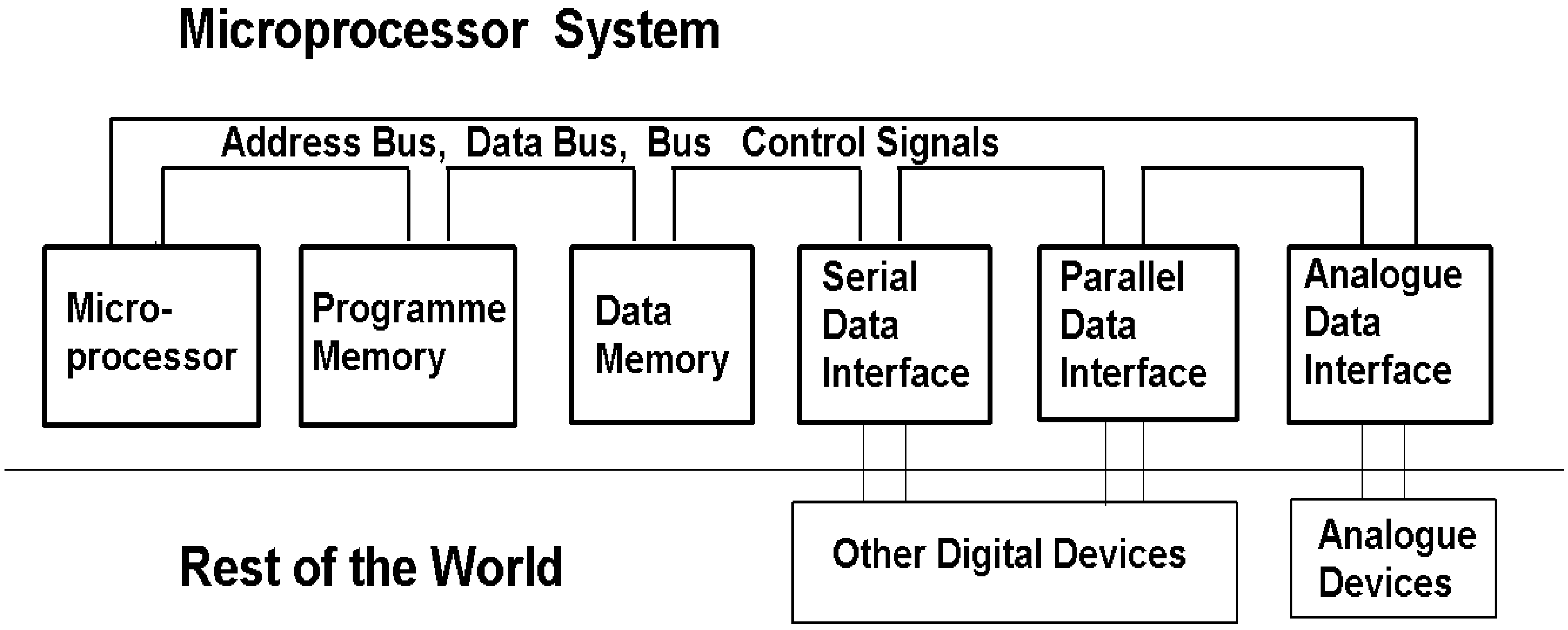
Typically Microprocessor within an Embedded System: Where Hardware meets Software



- Principal function(s) controlled by microprocessor embedded within it
- Computer (microprocessor or microcontroller) hidden from view
- Purpose designed for particular application

  (PC is really a general purpose computing machine rather than an embedded system)
- Embedded computer takes input variables from controlled system
- Computes output variables to control system
- Sometimes autonomous, or sometimes interaction with user or sometimes interaction with other systems

The Microprocessor System Overview



In this course the 68000 or 68HC000 processor is used to demonstrate aspects of the device hardware interface and software device access

Part 1.

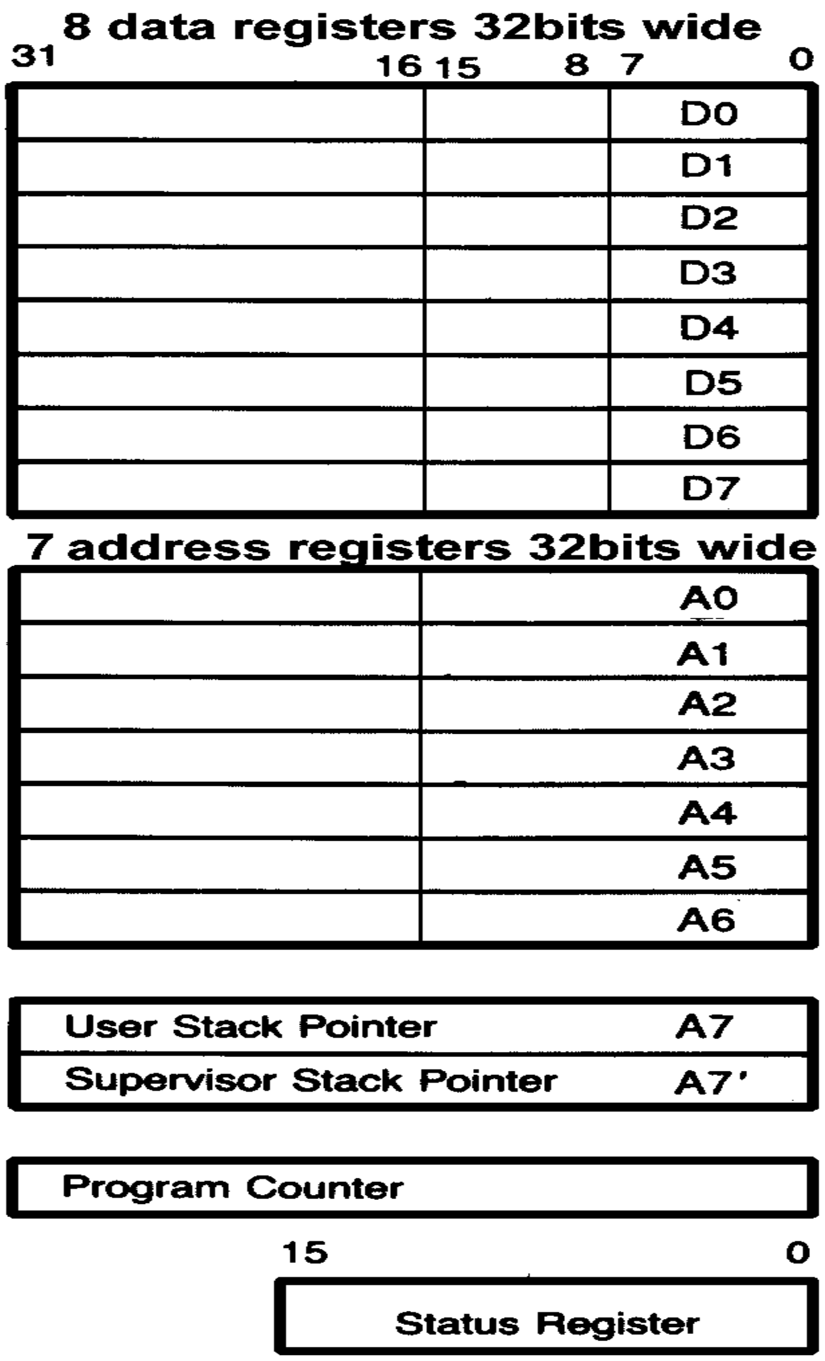
Review
Architecture & Programming of Microcomputer Systems:

- CPU architecture - 68000 example
- Programming model and instructions (reminder of 1st year)
- Microprocessor and the system bus
- Connection to memory & I/O devices
- Microcomputer organisation, signals and timing
- System architectures

68000 Example.

(or 68HC000)

Internal Software Program Model:



As a reminder of last year's microprocessor programming –

Quick review of instructions

follows….

Reminder: Move Instructions

A)MOVE General form:

MOVE.<data size> <source effective address>,<destination effective address>

Data size: B=Byte(8bits); W=Word(16bits); L=Long Word(32bits)

Some examples of types of addressing-

Register Direct :   MOVE.W D2,D3   moves lower 16bits D2→D3

Address Register Direct: MOVEA.W D3,A0 moves lower 16bits with sign extension

Address Register Indirect: MOVE.L (A1),D0   32bits from memory pointed to by A1-> D0

Address Register Indirect with Displacement: MOVE.<size> displacement16(An),Dn
MOVE.W $4(A0),D2   D2 ←memory at location given by (contents of A0 + 4)

Absolute: MOVE.L $C02E,D5 loads D5 with 32bit data word from location $FFC02E

Immediate: MOVE.L #$30,D2 loads D2 with immediate data (fills with leading zeros)

Address Register Indirect with Predecrement/Postincrement:
e.g.   MOVE.B –(A3),D3,
MOVE.L (A0)+,D4,
MOVE.W D4,(A2)+

Reminder: Arithmetic & Logic Instructions

B) Arithmetic Instructions

ADD.<data size> <ea>,Dn   Dn=Dn+<ea>
    and similarly for SUB (subtraction)

MULU <ea>,Dn   Dn ← <ea>lower word x Dn lower word
    similarly for DIVU (division)

C) Logical Instructions

ASL,   Arithmetic shift left   (lsb ←0)
ASR,   Arithmetic shift right (old msb→msb)
LSR,   Logical shift right (0→msb)
AND,   Logical AND
OR,   Logical OR
NOT,   all bits complemented 0←→1

Reminder: Programme Control Instructions

D) Program Control / Branch

BRA <relative address or label> unconditional jump in programme

JMP <ea> unconditional jump to location specified by effective address

B*cc* <relative address or label> conditional jump
where *cc* is flag condition.
e.g **BCC**=branch if carry clear, **BCS**=branch if carry set
Bcc often used after **CMP** – compare two data values

NOP no operation, time waster

E) Use of Stack / Subroutines

BSR, JSR unconditional branch/jump to subroutine (next programme address→stack)

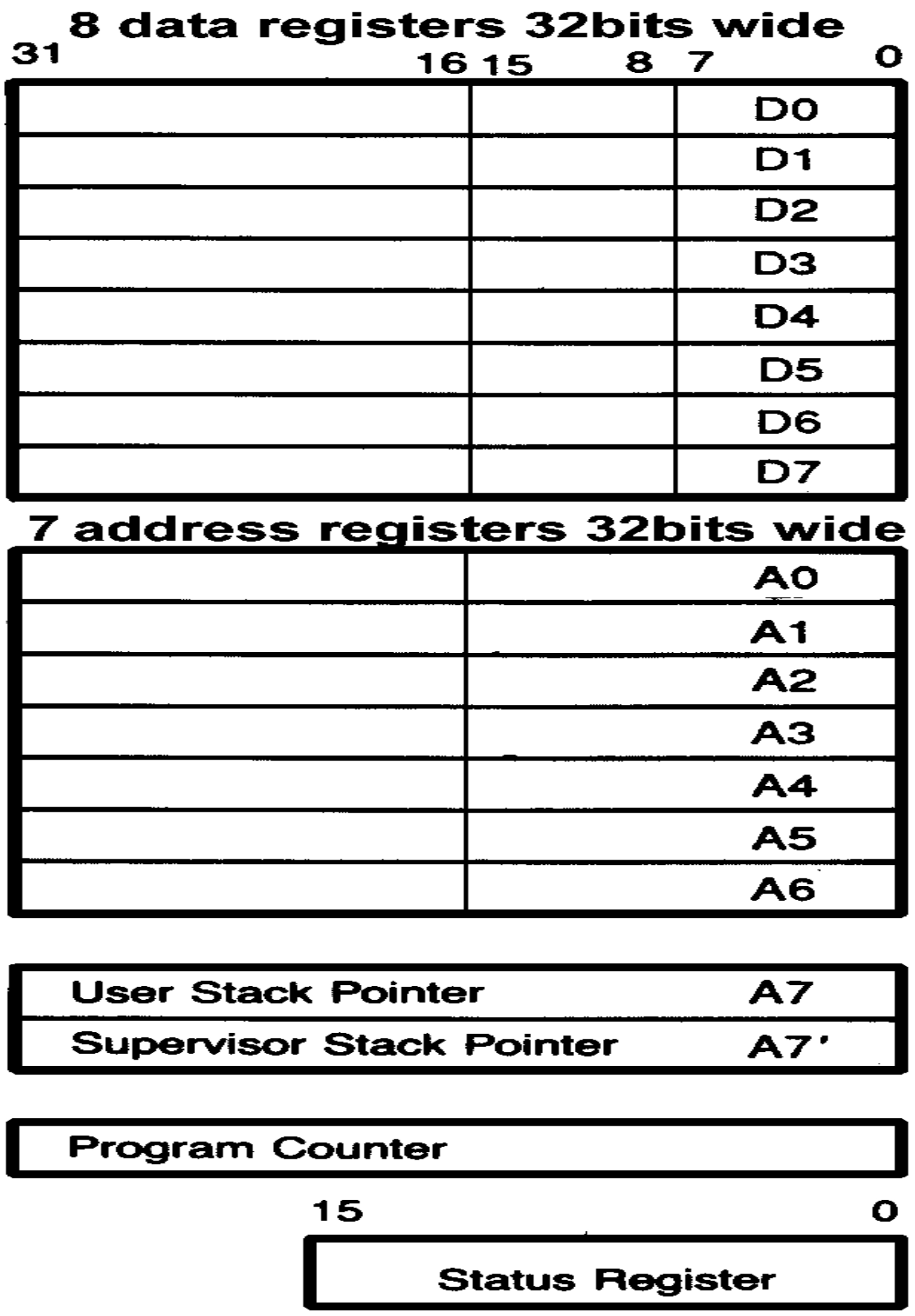
RTS Return from subroutine (changes programme counter to value previously saved on stack)

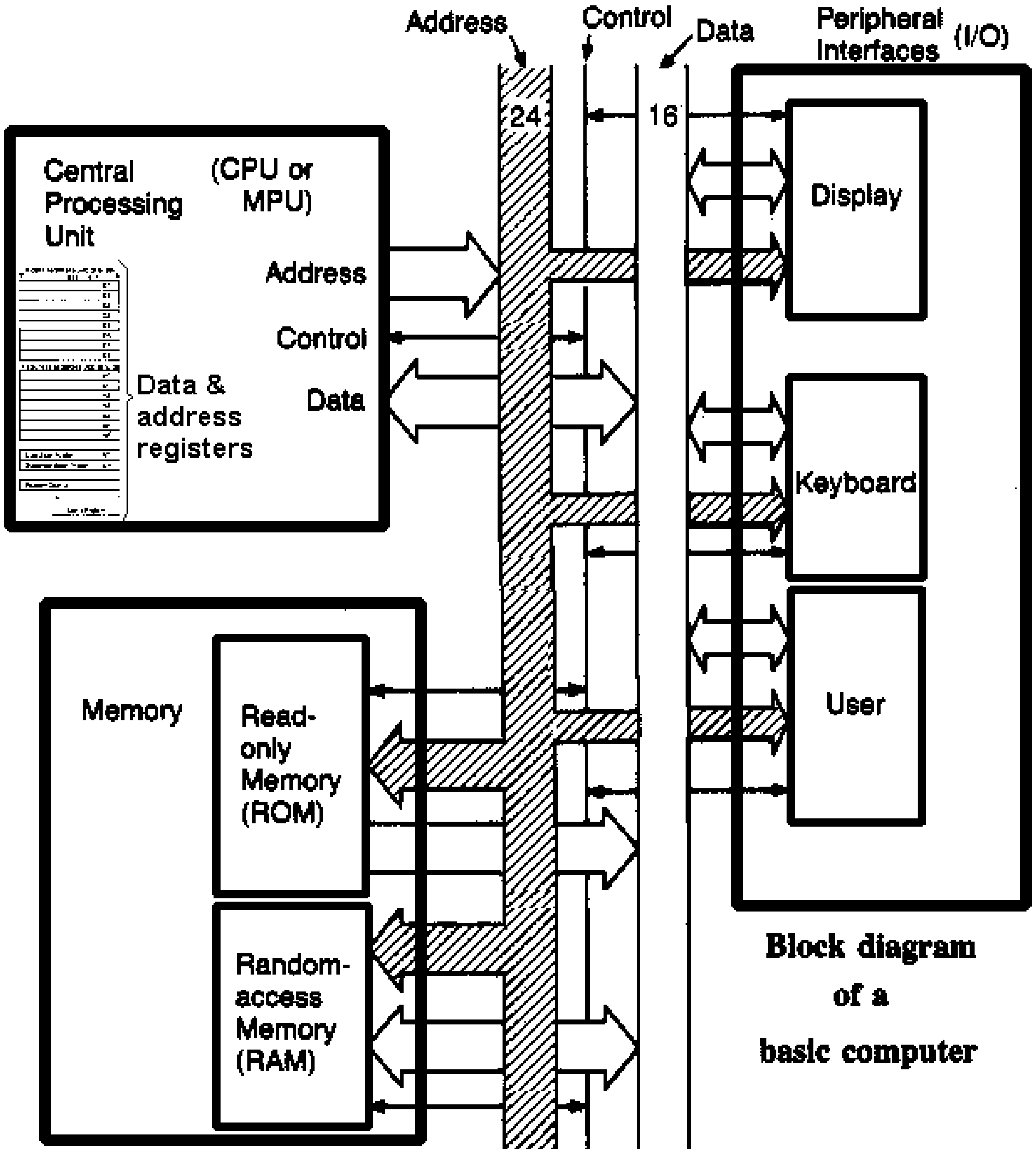
…..plus many other instructions.

Back to 68000 Programmer's Model →

Programme instructions intensively use the 8 data registers and 7 address registers in the CPU as intermediate data products or temporary variables in the course of processing data to / from the external world via external devices.

So where are these located relative to the typical system hardware? ……….





Block diagram of a basic computer

COMPUTER BLOCK DIAGRAM

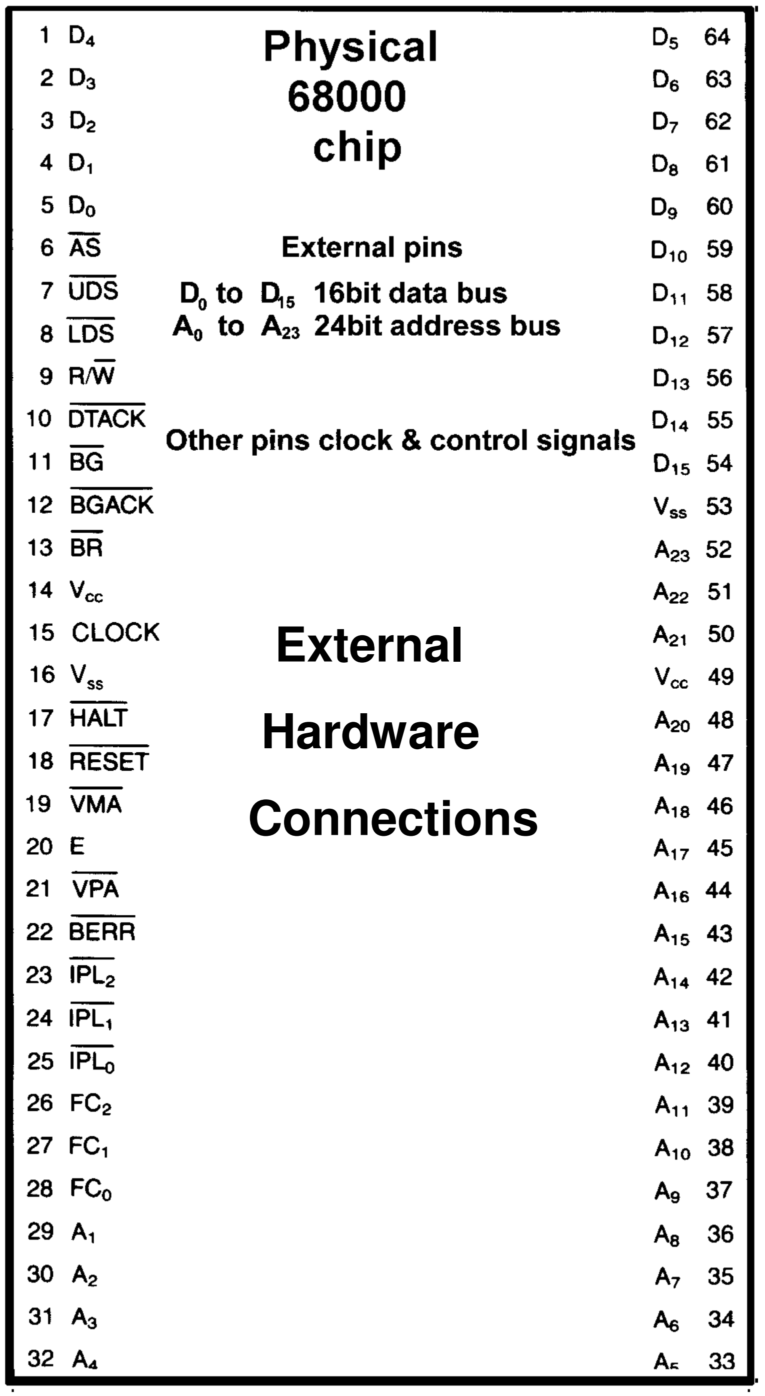
Each device connects to:
a) Data bus
b) Address bus
c) Control lines

Control lines determine:
- i) signals timing for correct operation
- ii) device selection/activation
- iii) data flow direction.

Only two devices allowed to communicate at any one time to avoid bus contention.

Thus data move operations mostly one of these 4 types:
i) Memory --> CPU or
ii) CPU --> Memory or
iii) I/O -->CPU or
iv) CPU -->I/O.

[Also a fifth type:
Direct Memory Access
I/O ←→ Memory
but requires special
DMA bus controller *see later*]



Summary of the 68000's 64 connection pins:

Vcc Voltage source (e.g. 5Volts above Vss)

Vss Ground

Clock: system clock input

Buses:

D_0 to D_{15} data bus lines - bidirectional

A_1 to A_{23} address bus lines, O/P

Main Control lines:

AS: Address strobe- valid address on A_1-A_{23}, O/P

R/W: direction of D_0-D_{15} bus,1=read,0=write, O/P

UDS,LDS: upper/lower data strobe [A_0], O/P

[effectively A_0:maps 8bit wide memories to 16 bits]

DTACK: Data Transfer Acknowledge, I/P

slower external devices can cause CPU to wait

RESET: resets CPU programme counter I/P

HALT: halts operation(I/P) or indicates failure(O/P)

IPL0-2: Interrupt request lines I/P

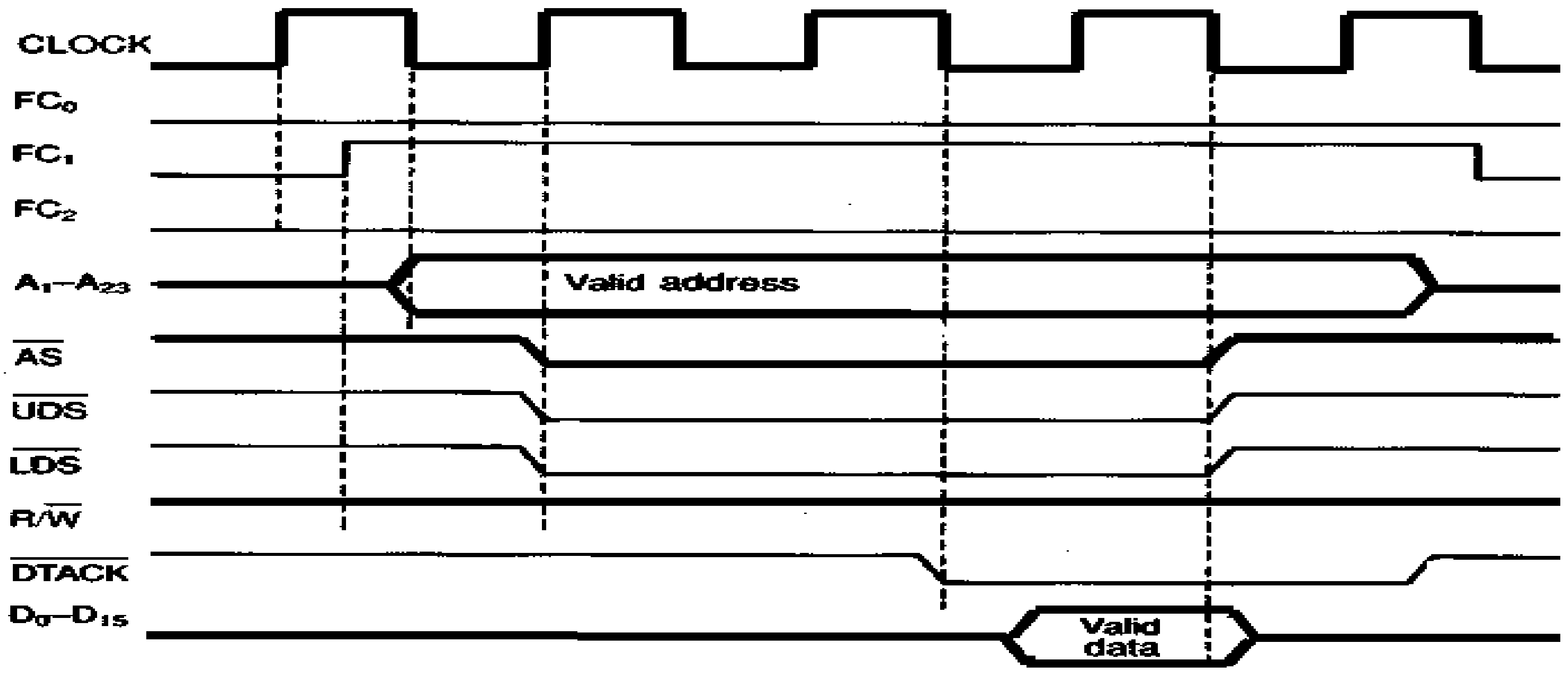
Others:

BR,BG,BGACK: for external DMA control of bus

FC0-FC2:monitor: programme, data, interrupt ,O/P

E,VMA,VPA,BERR extra signals for interfacing

Programme Memory Read Cycle (Main signals in bold)



Main aspects: FC0-FC2=010 indicates program opcode fetch (alternative 001 for data)

Valid address →A_1-A_{23}, UDS,LDS=00 means16bit read (10=d_{0-7}, 01=d_{8-15} only)

Address Strobe, **AS** allows address bus to be decoded for memory chip select

R/~W stays high throughout as this is a read operation

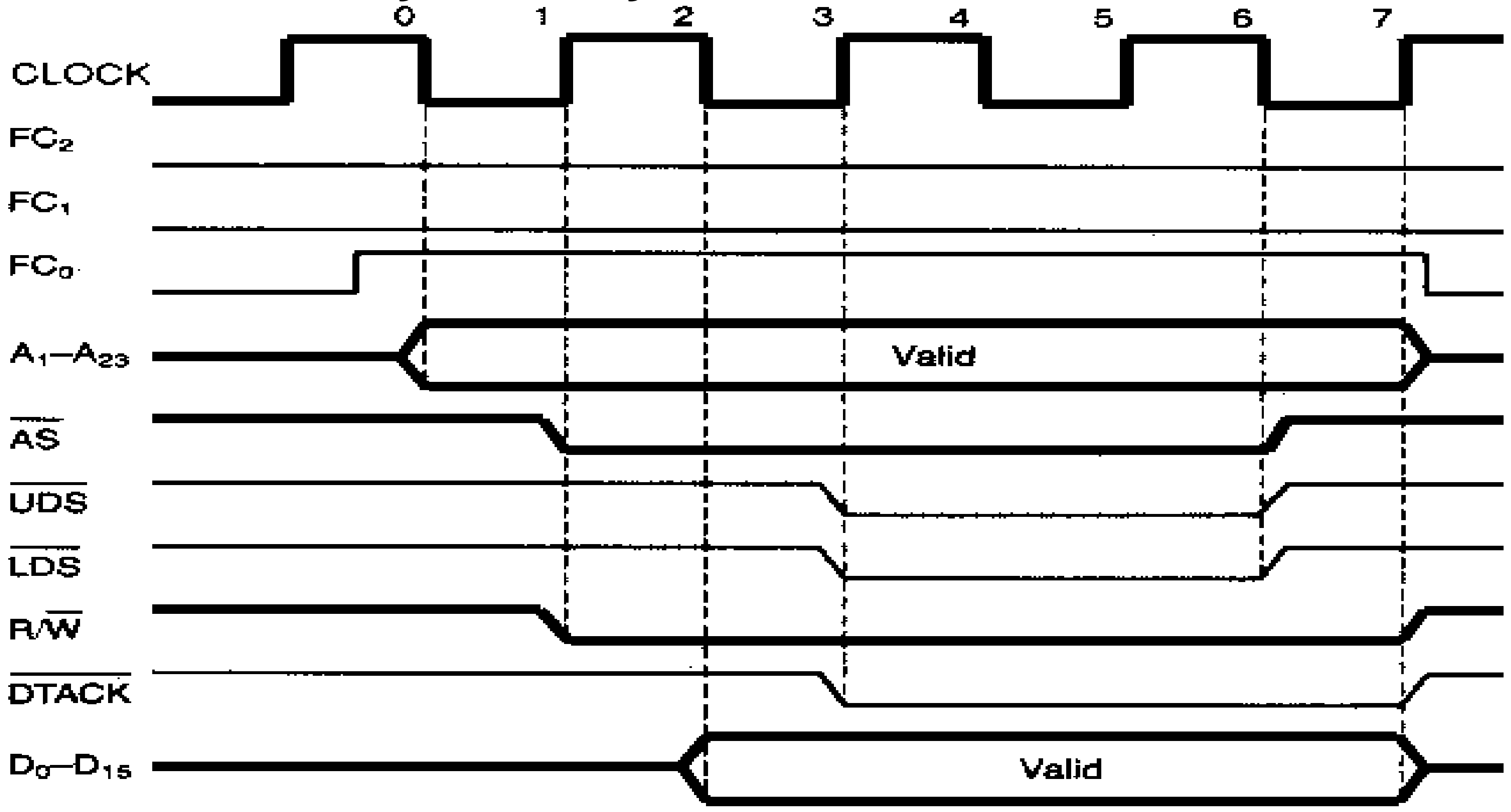
External device/address decode asserts ~DTACK as data placed on bus

If memory device is slow ~DTACK assertion can be delayed to provide wait states

D_0-D_{15} Data bus receives valid data from addressed memory before AS returns.

68000 uses a 2-word prefetch, absorbing program fetch cycles within execution cycles

Memory Write Cycle (main signals in bold)



Main aspects: FC0-FC2=001 data memory.

Valid address →A_1-A_{23}, UDS,LDS=00 means16bit write (10=d_{0-7}, 01=d_{8-15} only).

Address Strobe, AS allows address bus to be decoded for memory chip select.

R/~W goes low to indicate this is a write operation,writing **D_0-D_{15}** to memory.

External device/address decode asserts ~DTACK as device reads data from bus.

If memory device slow ~DTACK assertion can be delayed to provide wait states.

System Design

1) Before Designing system decide on requirements:

- Amount of programme memory (ROM)
- Amount of read/write data memory (RAM)
- Number & type of I/O ports
- Other system and peripheral components as needed

2) Software must be considered.

3) Then individual component types chosen, considering their characteristics (timing, voltage levels,etc) & requirements

4) Circuit wiring, board design & board layout completed

Part 2.
Programme & Data Storage

- Types of memory device

- Connecting memory to the processor

- Memory device address decoding

Types of Memory

Random Access Memory, RAM (data volatile- lost on power off)

RAM used for data, can be written to & read from

- **Static RAM – each bit stored in simple circuit of a few transistors, e.g. flip-flop**
- **Dynamic RAM- each bit stored as charge on a single transistor gate but needs refresh circuitry as gate is a *leaky* capacitor and data lost otherwise**

SRAM faster, takes more power, less dense → expensive, but easy to use

DRAM simpler, lower power, cheaper, requires extra refresh control, more complex to use.

Read Only Memory, ROM (data non-volatile, remains after power cycling)

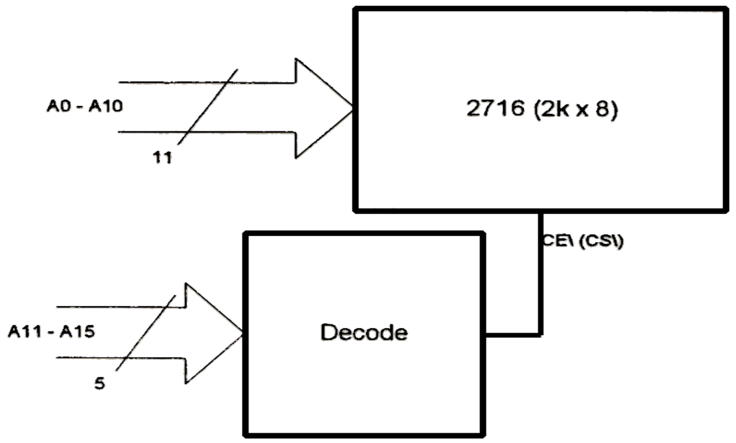
ROM data remains after power off.

- **Mask programmed – custom written at manufacture, e.g. PC boot up programme**
- **PROMS – semi-custom- written only once to chip by specialist equipment/co**

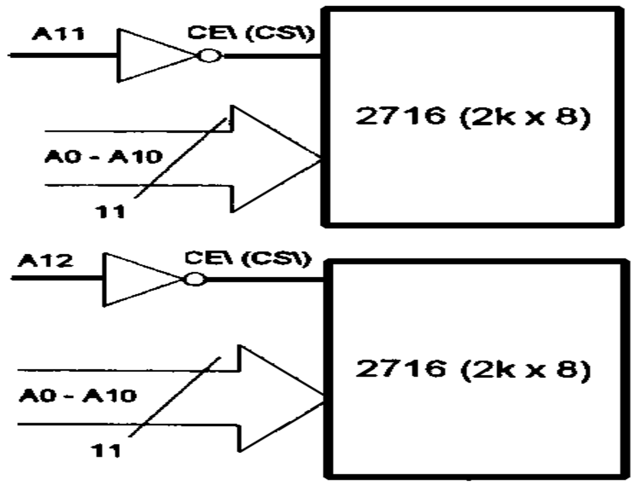
  data 0/1 stored as fuses blown/unblown or as OTP (see below)
- **EPROM – user programmed by EPROM programmer. Data stored as charge on high impedance gates- can be erased by ultra-violet light through window in chip & reprogrammed.**

  One time programmable, OTP, = version of EPROM chip without window
- **EEPROM- similar to EPROM but erased electrically without being removed from circuit. Erased in blocks of memory – in system programmable**
- **Flash memory, similar but simpler→ very dense memory (silicon hard disc)**
- **FRAM access as fast as RAM but data non-volatile**

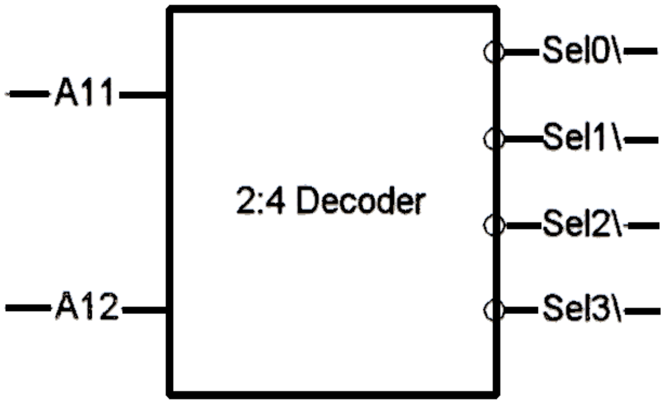
Address Decoding



a) General address decoding
Chip selected by specific combination of higher address line values.

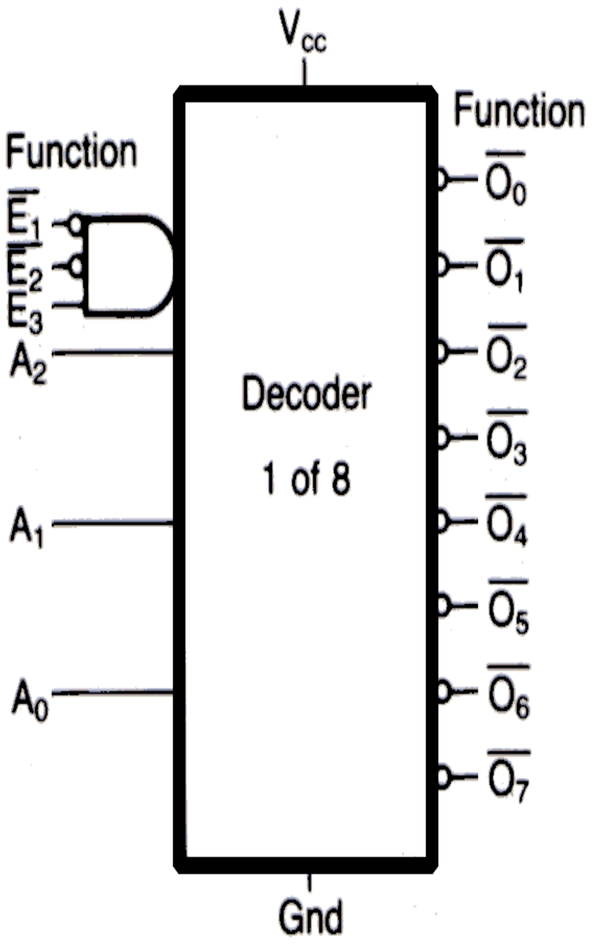


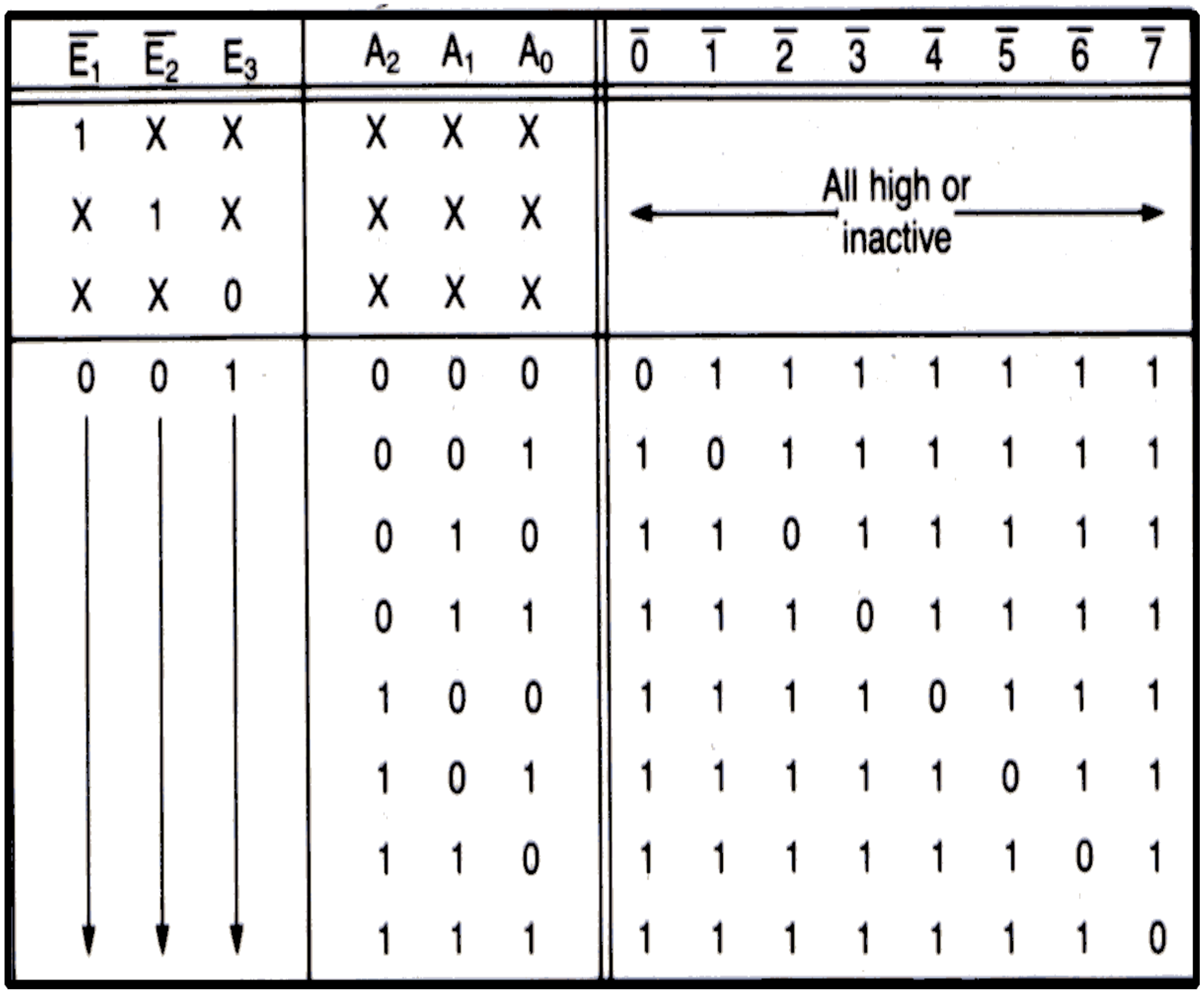
b) Linear Address Decoding
Each chip select uses a dedicated address line- simple for small systems but wasteful and can lead to bus contention (>1 device selected at once!! e.g. A11 & A12 must not both =1)



c) Full Address Decoding
Logic used to provide a maximum number of chip selects from address lines.
E.g. two address lines A11 & A12 have four possibilities (00,01,10,11)→ each combination decoded for a chip select.

Full Address Decoding: Decoder Chip 74138





$\overline{E_1}$	$\overline{E_2}$	E_3	A_2	A_1	A_0	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$
1	X	X	X	X	X	All high or inactive							
X	1	X	X	X	X								
X	X	0	X	X	X								
0	0	1	0	0	0	0	1	1	1	1	1	1	1
↓	↓	↓	0	0	1	1	0	1	1	1	1	1	1
			0	1	0	1	1	0	1	1	1	1	1
			0	1	1	1	1	1	0	1	1	1	1
			1	0	0	1	1	1	1	0	1	1	1
			1	0	1	1	1	1	1	1	0	1	1
			1	1	0	1	1	1	1	1	1	0	1
			1	1	1	1	1	1	1	1	1	1	0

When chip is not enabled: all 8 outputs high independent of A inputs
When chip enabled (~E1,~E2,E3=001) only one output goes low, rest high
Inputs A1,A2,A3 select which of 8 outputs goes low

Programme memory : ROM/EPROM Two examples:

M6836 16k x 8 Byte wide

Data: DQ_0-DQ_7

Address: A_0-A_{13}

~G is Read

~E chip select

Intel 27210 64k x 16 16bit word size

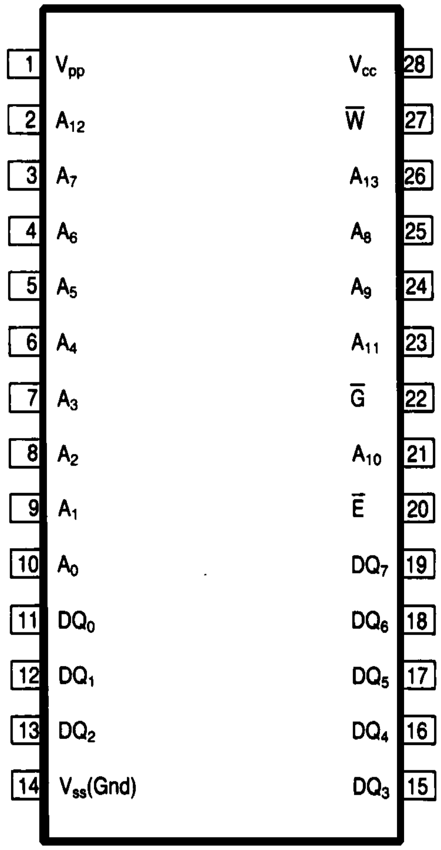
Data: O_0-O_{15}

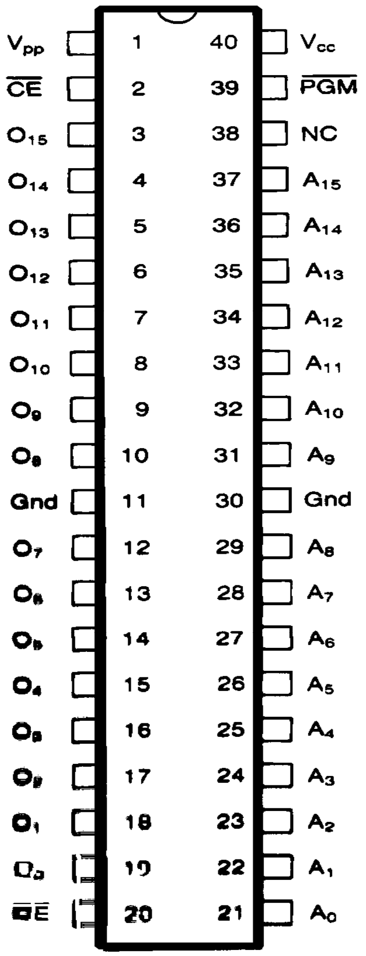
Address: A_0-A_{15}

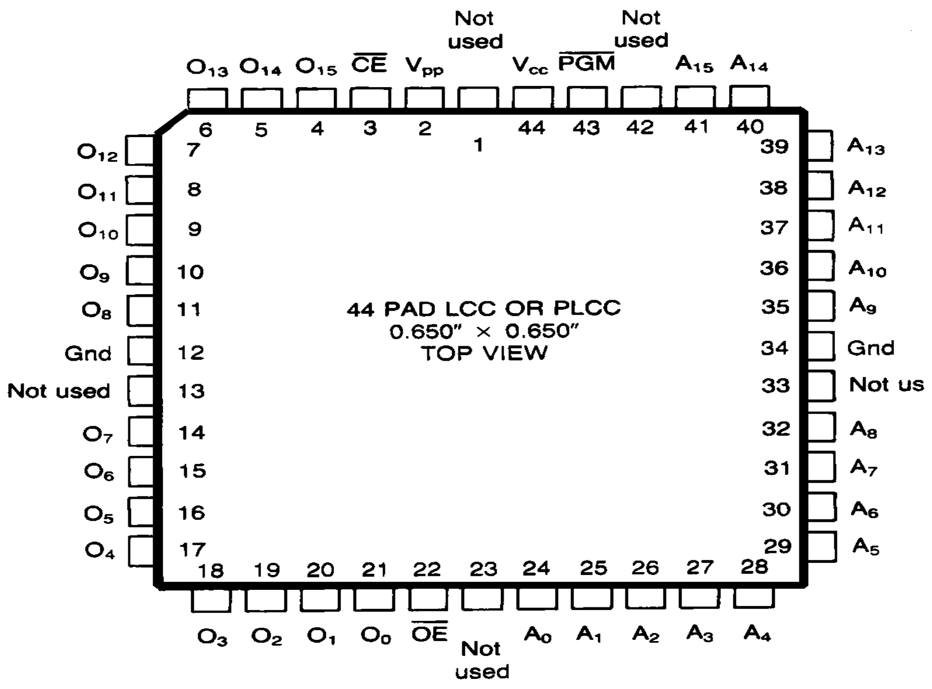
~CE is chip select

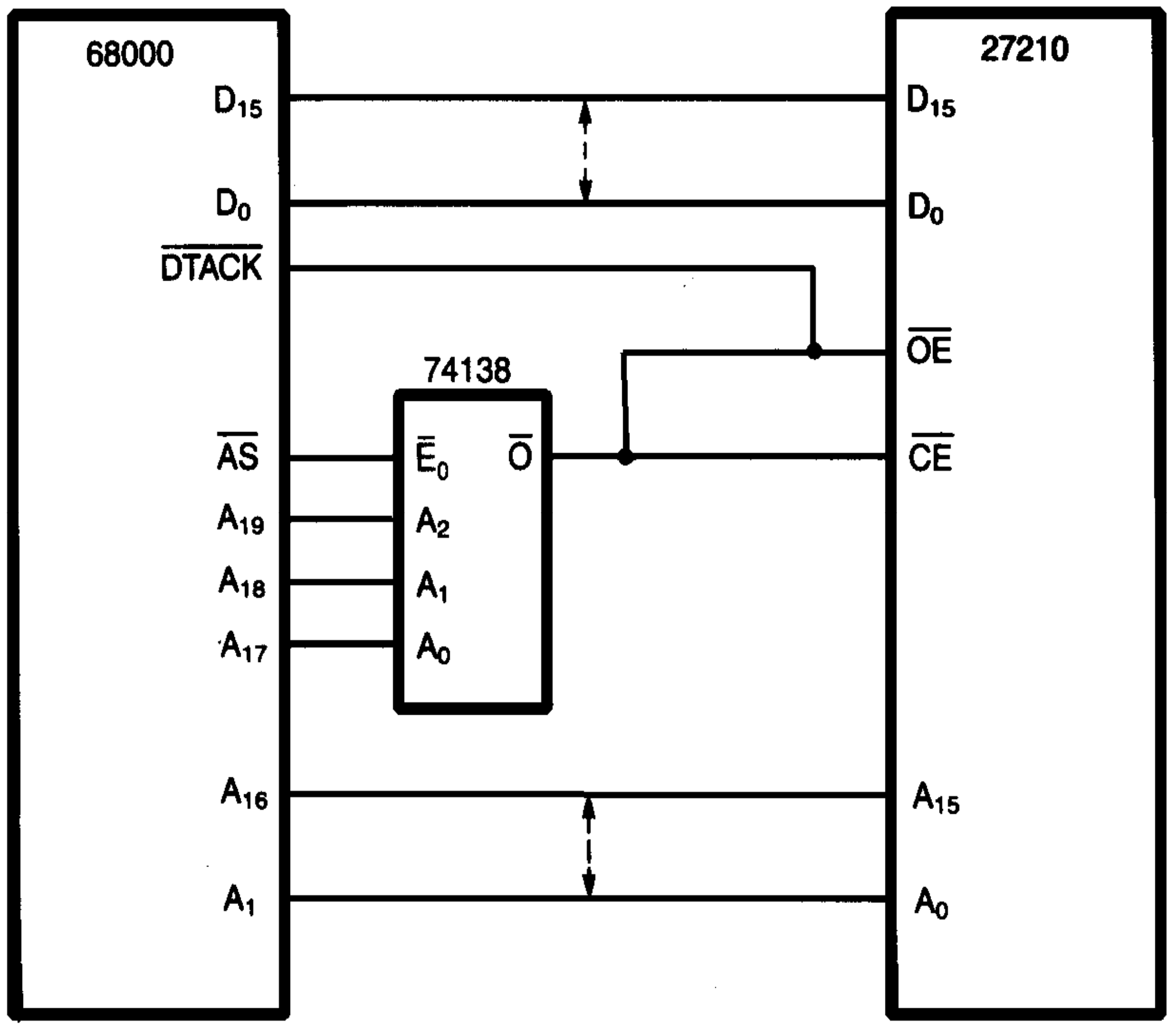
~OE is enable ouput (read data from ROM)

~PGM for programming data into ROM





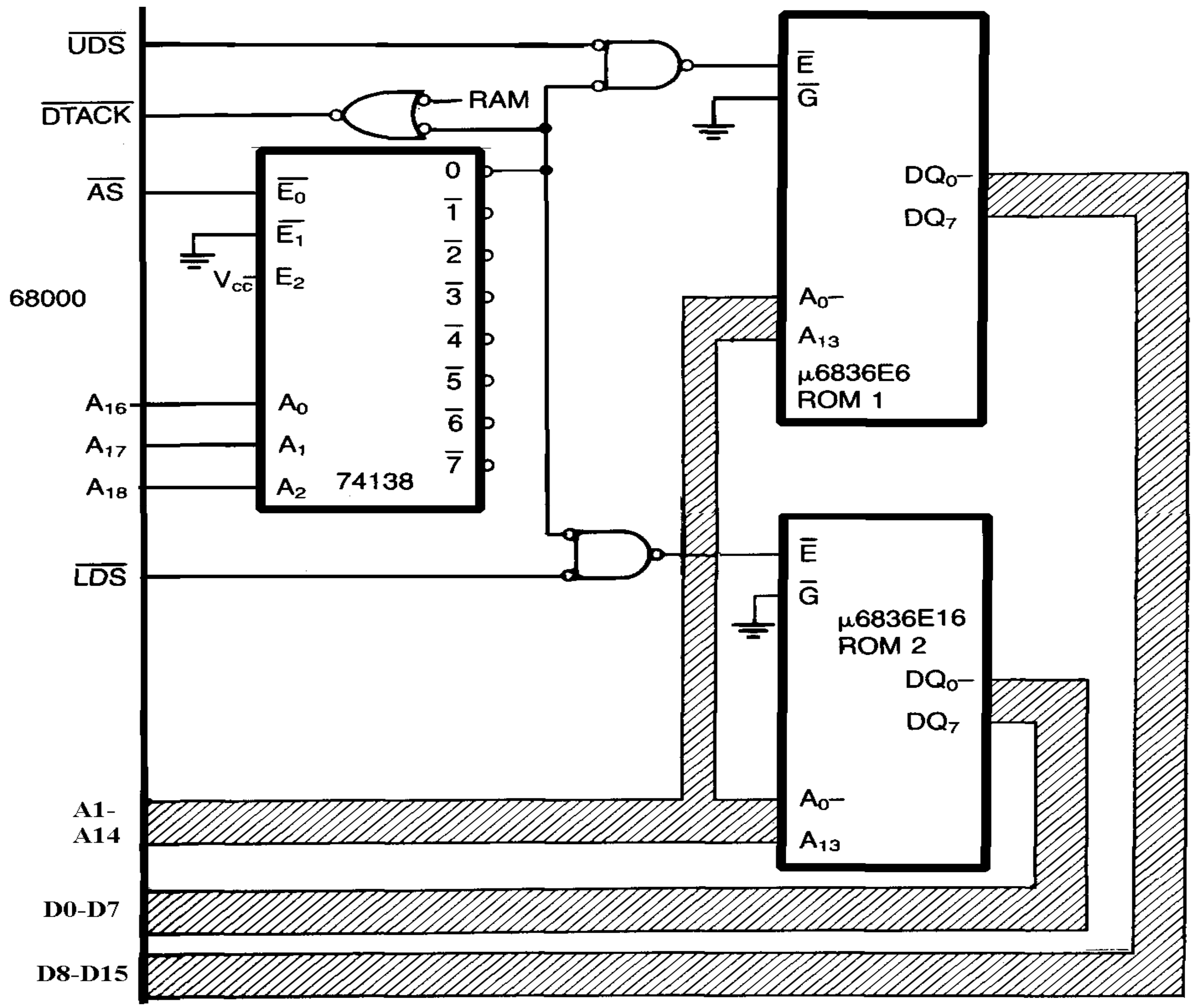




16 bit wide ROM:

- D_0-D_{15} ROM→uP
- All uP A_1-A_{16}
  → ROM A_0-A_{15}
- 64k x 16 ROM
- ~CE from 138 decoder when
  A_{17},A_{18},A_{19}=000
  Other combinations for other devices
- As ROM all accesses are read so ~OE=~CE
- ~DTACK low
  while ROM selected

A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1		
0	**0**	**0**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**ROM**	**00000H-0FFFFH**
0	**0**	**1**	**……………………………………………………………………..**																**Next Device**	**10000H→**
0	**1**	**0**	**…………………………………………………………………….**																**Another Device**	**20000H→**

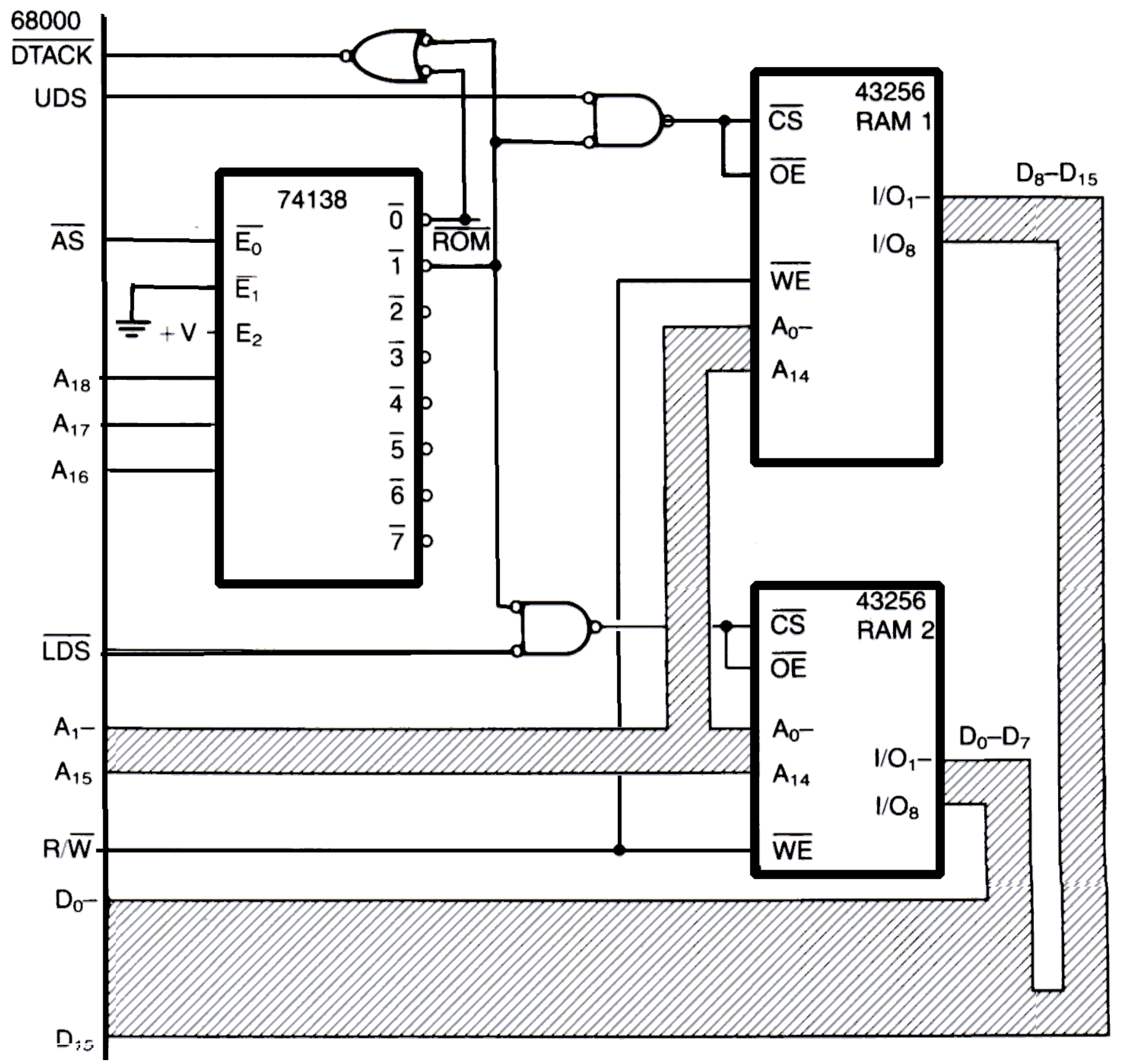


Using two 8 bit wide ROMs for 16bits data bus

- A_0= UDS/LDS
- A_1-A_{14} micro→ both ROMs A_0-A_{13}
- ROM1 = D_8-D_{15}
- ROM2 = D_0-D_7
- Address decoder selects ROMs for A_{16}-A_{18}=0 ROM: 0000H-03FFFH
- other 138 outputs used for other devices, RAM etc

ADDRESS BUS																								
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DEVICE
					0	0	0		X	X	X	X	X	X	X	X	X	X	X	X	X	X	U	ROM 1
					0	0	0		X	X	X	X	X	X	X	X	X	X	X	X	X	X	L	ROM 2

U = $\overline{\text{UDS}}$ L = $\overline{\text{LDS}}$ X = Variable



Connecting RAM

- Addition of two 32K x 8 RAM to previous slide
  (two of ROM of last slide not shown for clarity)
- Again pair for 16bits wide
- ROM A_{16}-A_{18}=000
- RAM A_{16}-A_{18}=001
- Now R/~W needed
- ~DTACK as long as either ROM or RAM accessed.

18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Chip
0	0	0		X	X	X	X	X	X	X	X	X	X	X	X	X	X	U	ROM 1
0	0	0		X	X	X	X	X	X	X	X	X	X	X	X	X	X	L	ROM 2
0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	U	RAM 1
0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	L	RAM 2

ROM 0000-3FFFH

RAM 8000-FFFFH

Part 3.
Parallel I/O and peripheral devices:

- Buffers and latches

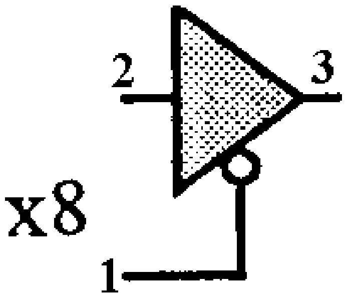
- Example input and output devices

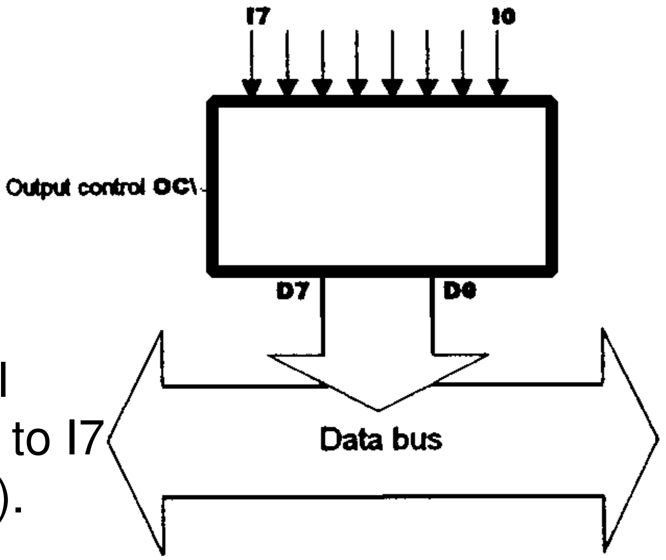
- Programmable I/O devices

- Counter-timers

Buffers for digital input port:

Buffers enable to pass at specific times.
Single buffer: pin 1 when low passes data from pin 2 to pin3, otherwise high impedance on pin3.

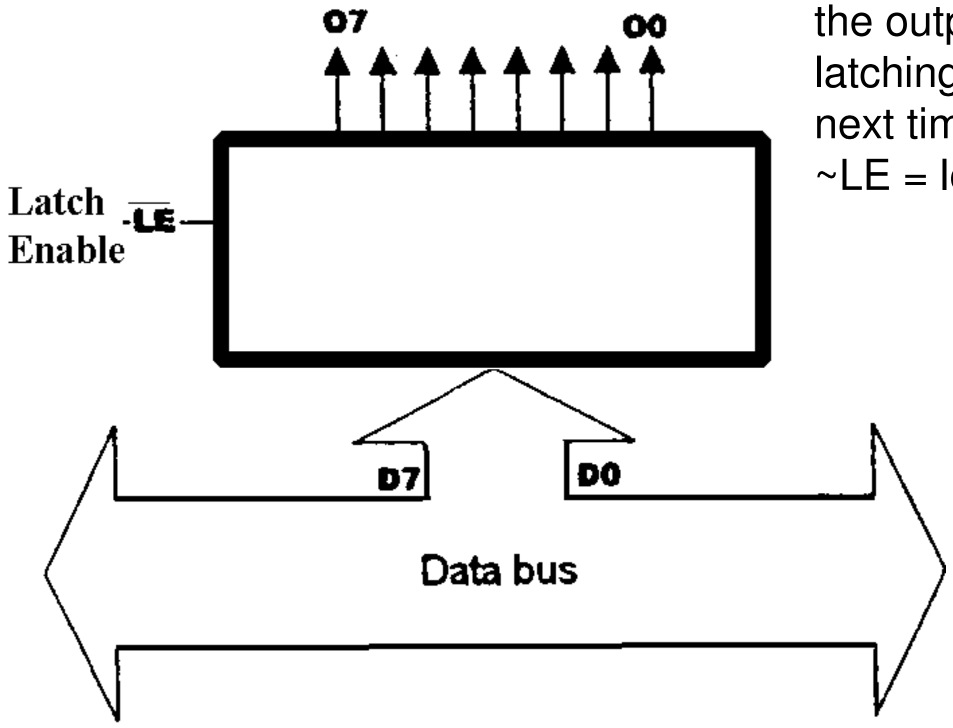


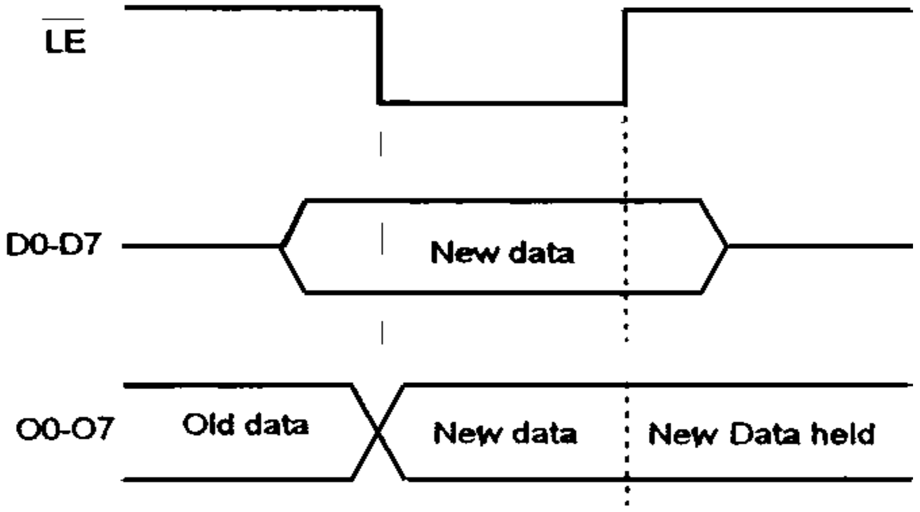


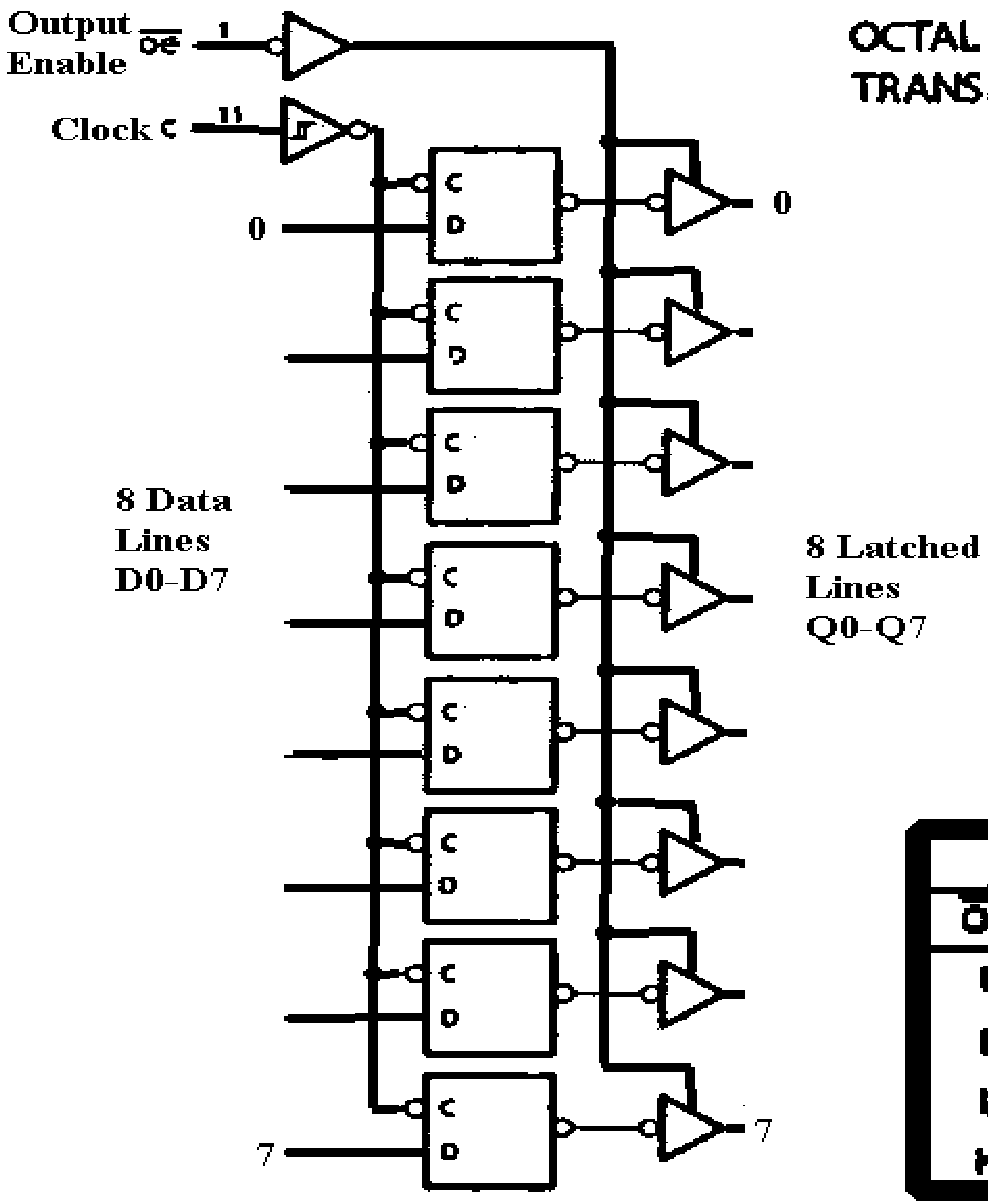
Combining 8 buffers in parallel provides a means for digital input. When common output control low, data from lines I0 to I7 is passed onto data bus for microprocessor to use (INPUT).
~Output control = logical OR of ~PortCS, ~Bus Read

Latches for digital output port:

D-type latches used to sample & hold data - latch data. Data on the bus, e.g.from the microprocessor, sent to the output port. Address decoding for port enables latch, latching data at end of pulse. Port outputs new data until next time port addressed (OUTPUT).
~LE = logical OR of ~Port CS, ~Bus Write)







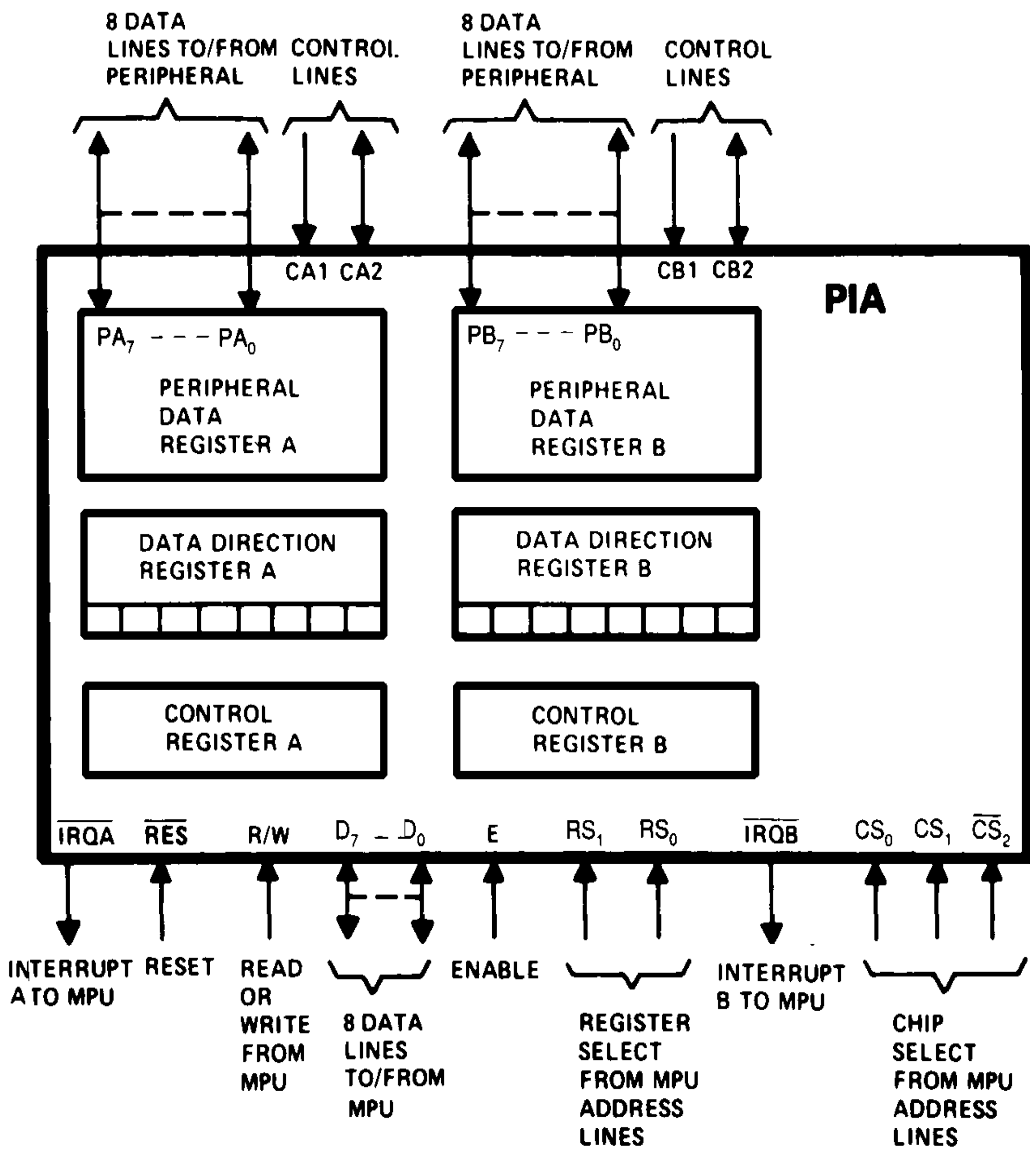
OCTAL D-TYPE TRANSPARENT LATCHES

Octal latch can be 8bit Port

Function Tables

INPUTS			OUTPUT
$\overline{OE}$	C	D	Q
L	H	H	H
L	H	L	L
L	L	X	Q_0
H	X	X	Z

Peripheral Interface Adaptor(PIA)



Example of MC6821 (simple 8 bit port)

Two Ports: A & B

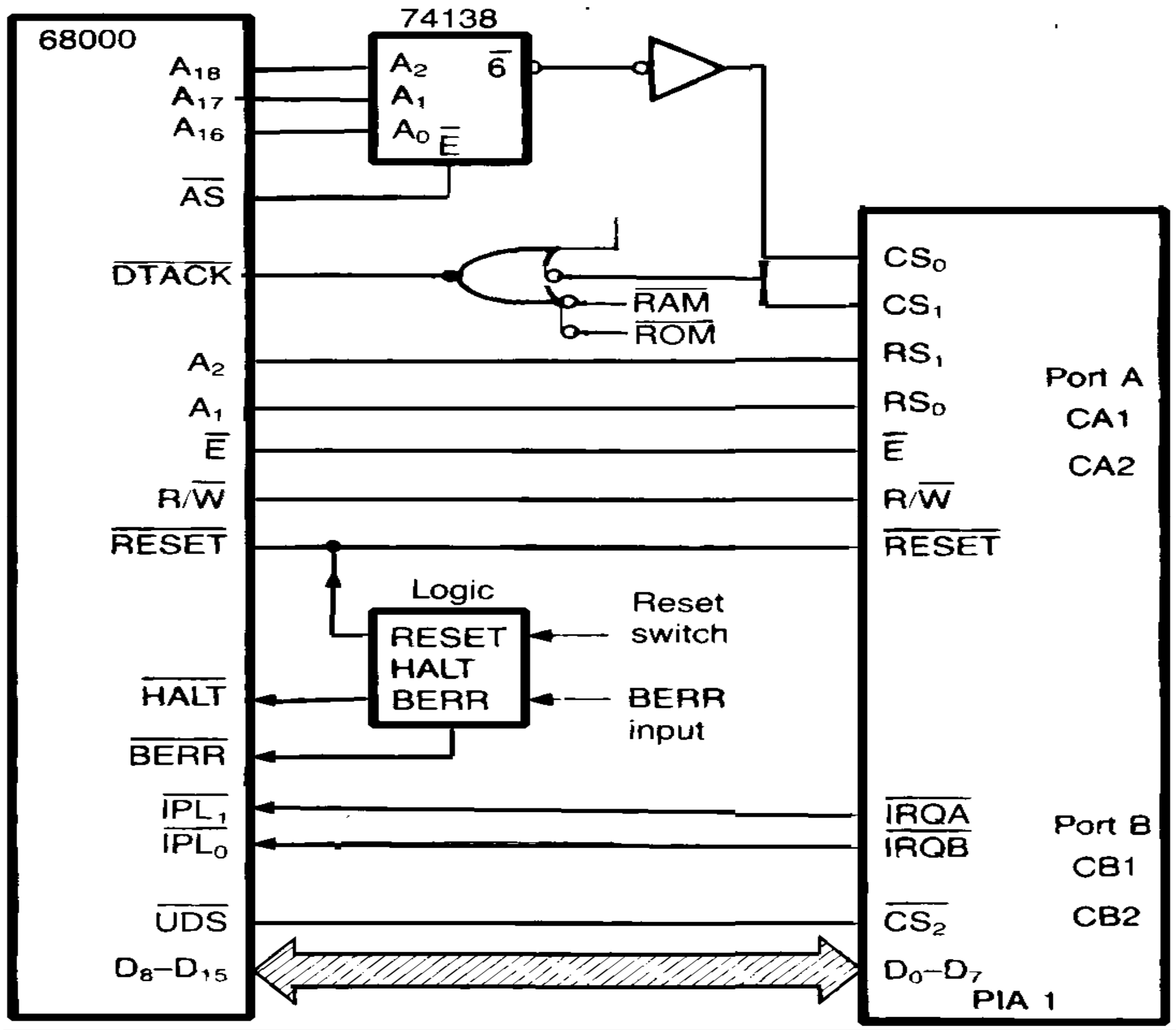
Each port has 3 registers:

1)Peripheral Data Register
buffers actual port data

2)Data Direction Register
each bit 0 for I/P, 1 for O/P

3)Control Register
sets condition for data flow, interrupt, & handshake with external devices

Connecting the PIA



Connecting to previous system of ROM & RAM, using decoder set for A_{16}-A_{18}=110 and A_0=0 for port.

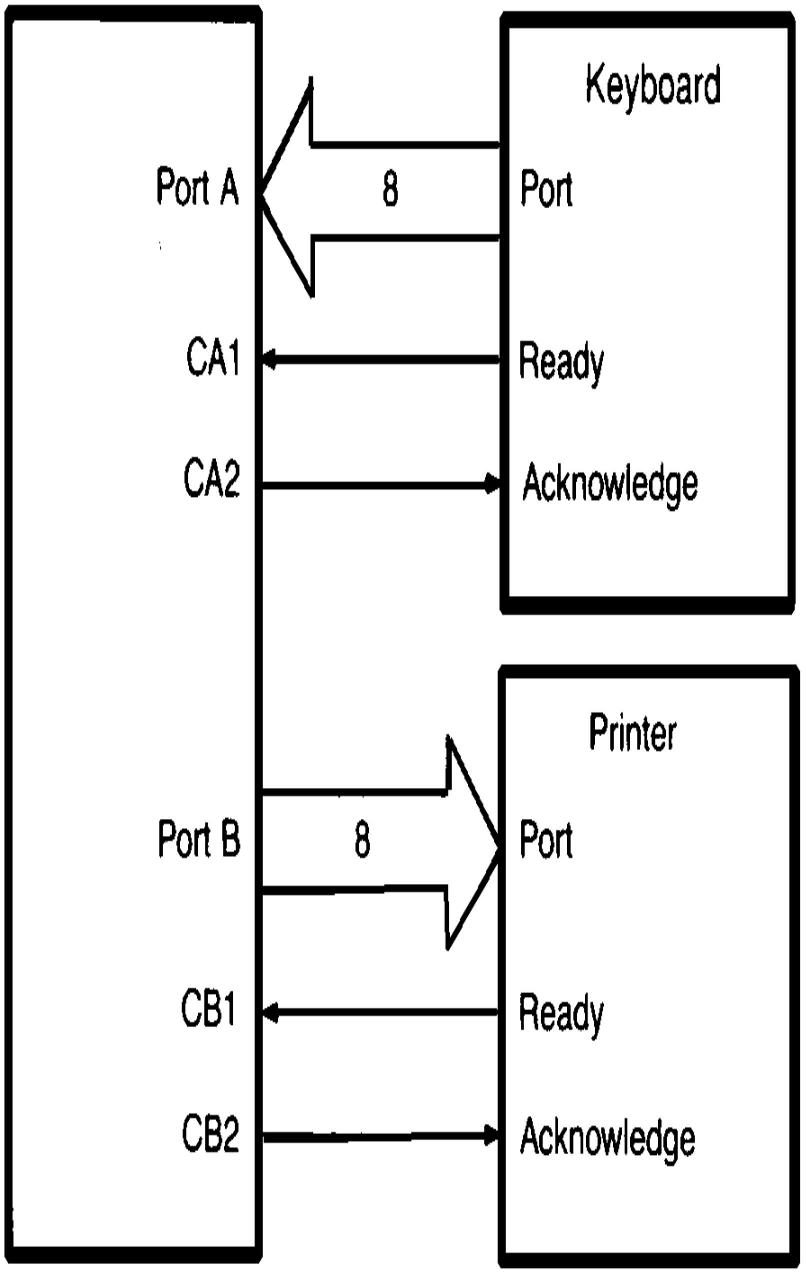
Lower address bits A_1, A_2 select PIA registers for I/O data flow Control.

Interrupt requests connected to 68000 interrupts so that each Port A & B can be processed by Separate interrupt routines.

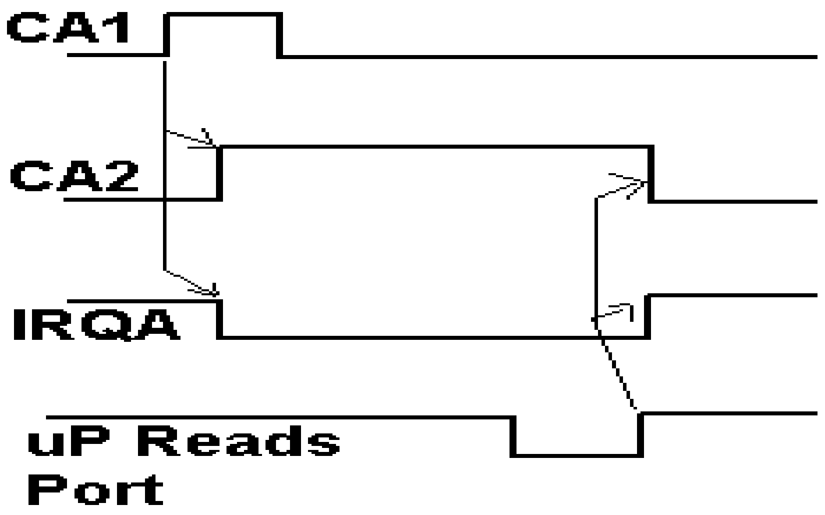
Address Lines																			
18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Chip
0	0	0		X	X	X	X	X	X	X	X	X	X	X	X	X	X	U	ROM 1
0	0	0		X	X	X	X	X	X	X	X	X	X	X	X	X	X	L	ROM 2
0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	U	RAM 1
0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	L	RAM 2
1	1	0													0	R	S	U	PIA 1

U = $\overline{UDS}$ L = $\overline{LDS}$ R = RS_1 S = RS_0

PIO Handshake with external world



Input- keyboard example:

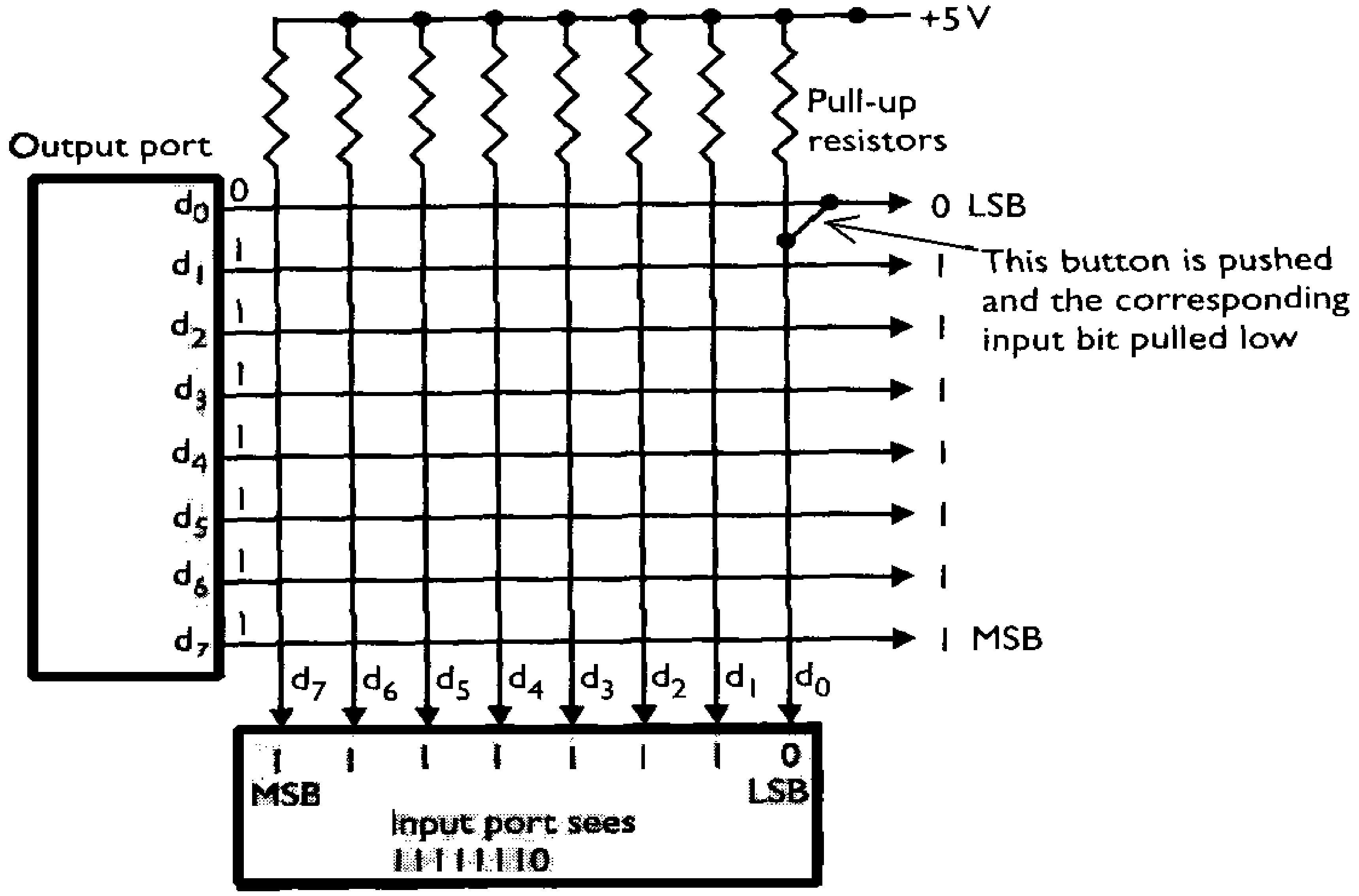


Keyboard Key pressed → CA1→high
Keyboard tells PIA that data ready to be read. CA1 used to strobe data into PIA data register.

PIA acknowledges keyboard by CA2→high and at same time tells uP by setting IRQA →low

Microprocessor services interrupt request and reads PIA data register. Act of reading resets IRQA →high and resets CA2→low telling keyboard that it is ready for more data.

Keyboard Encoder Example:

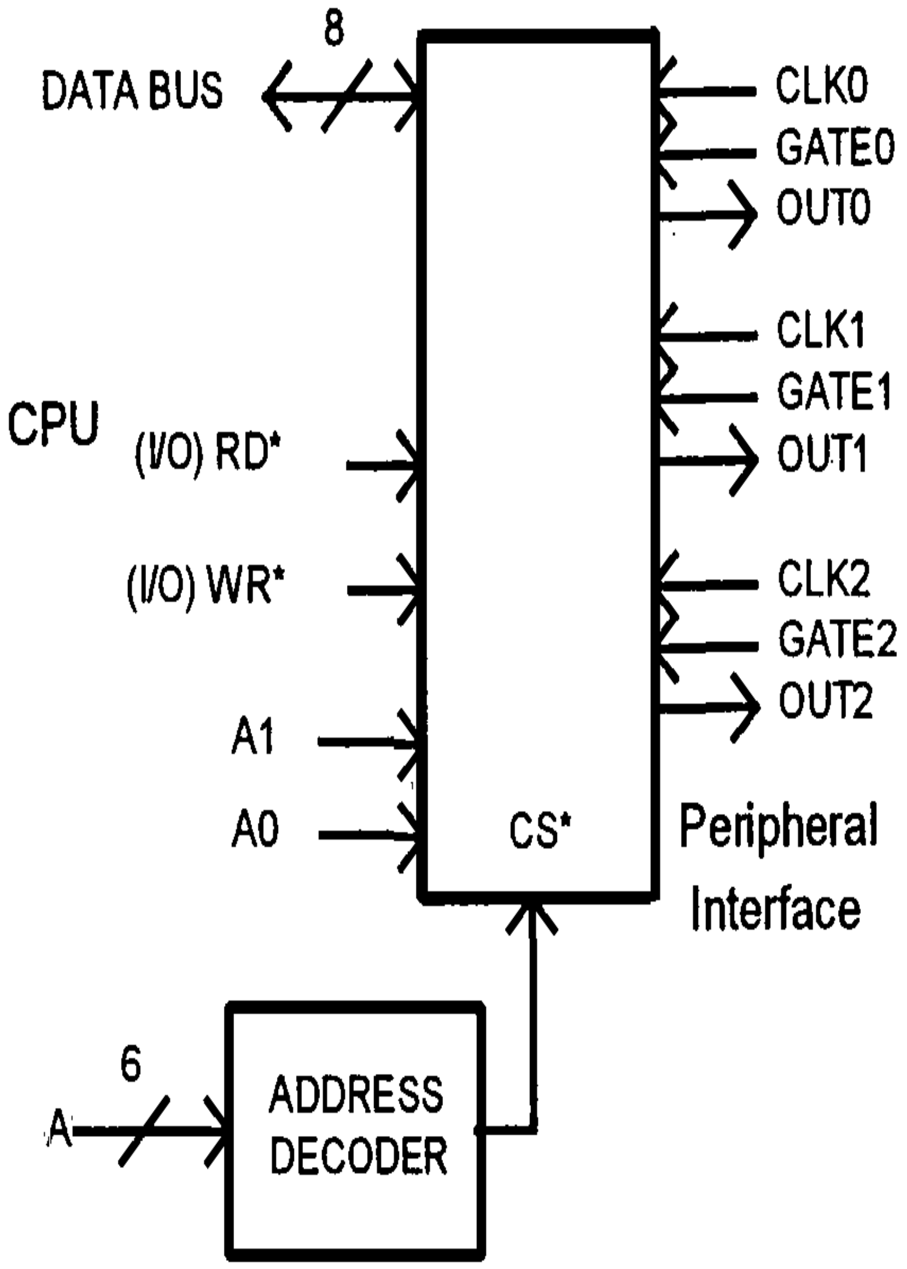


Rows scanned with travelling '0' on output port until keypress causes input <11111111
Then key identified by combination of known position of '0' on O/P port
and measured position of '0' on I/P port.

Code for keyboard scanner/reader

```
        ORG $002000
Xlines  EQU     $008000         ;Output port for rows
Ylines  EQU     $008002         ;Input port for columns
        MOVE.B  #%01111111,D0   ;Initial X value with ‘0’
        MOVE.B #-1,D1           ;Preset X counter = -1
XLOOP   ROL.B #1,D0             ;Rotate position of ‘0’
        ADD.B #1,D1             ;Increment X counter
        AND.B #%00000111,D1     ;X counter is modulo 8 (values 0-7 only)
        MOVE.B D0,Xlines        ;Output X value to keyboard
        MOVE.B Ylines,D2        ;Read Y value from Keyboard to D2
        CMP.B #%11111111,D2     ;Any ‘0’ in Y – Any key pressed?
        BEQ     XLOOP           ;Repeat until key pressed
        CLR.B   D3              ;Preset Y counter = 0
YLOOP   CMP.B #%11111110,D2     ;test for a ‘0’ in lsbit of D2
        BEQ JOIN                ;Exit to concatenate X & Y values
        ROR.B  #1,D2            ;Rotate D2 one place right
        ADD.B #1,D3             ;update Y counter
        BRA YLOOP               ;test next bit position for ‘0’
JOIN    LSL.B #3,D3             ;Shift Y counter 3 places to the left
        OR.B D3,D1              ;Add in X counter
        RTS                     ;Return value in bits: 00yyyxxx   ( 0-63₁₀ )
```

Counter-Timer Chips



This example three separate counter-timers
Each has clock input, a gate input, and an output:

a)Clock can be supplied from the micro-processor clock, or by an external system.

b) Gate is a signal that enables/disables counting

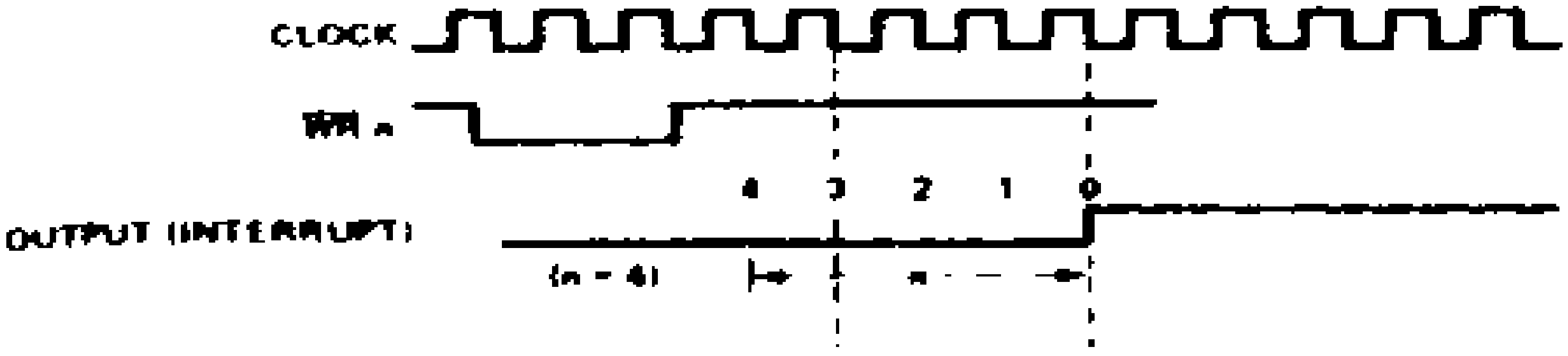
c) The output is changed when the counter reaches a preset value, counted down$\rightarrow$0.

Uses:

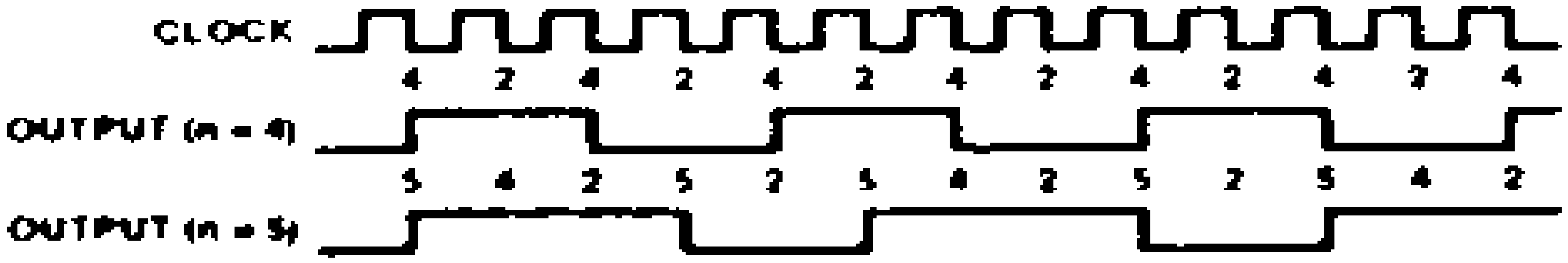
- Output can be used as interrupt to uP
- Enables accurate time delays to be generated under software control
- Multi-Mode configured by software
- Used for delay instead of software timing loops- frees up uP to do other tasks
- Can be used to count external events
- Watchdog timer- unless software reloads counter before an initial long count value reaches zero$\rightarrow$ resets system. Checks against ‘endless loop’ type software hangups - ensures continued operation of essential systems.

Examples of two of the many Modes provided by a Counter/Timer Chip:

Mode 0: Interrupt on Terminal Count



Mode 3: Square wave generator



Other modes:
Mode 1- Programmable One-Shot, **Mode 2- Rate generator**
Mode 4- Software Triggered Strobe, **Mode 5- Hardware Triggered Strobe**

Part 4.
Interrupts:

- Need for interrupts
- Principles of interrupt-driven I/O
- Interrupt programming techniques
- Interrupt Priority & Interrupt Vectors

Output to Port with fixed software delay (without interrupts)

```
Port     EQU     $01800 Location of Port
Count    EQU     128     Size of block to output
Deloop   EQU     64      wait loop
         ORG     $000400 Program origin
  :
         MOVE    #Count,D1    ;set up loop counter
         LEA     Table,A0     ;A0 points to table in memory
         LEA     Port,A1      ;A1 points to Port
  :
LOOP1    MOVE.B  (A0)+,D0     ;D0←memory([A0])
                              ; [A0]←[A0]+1
         MOVE.B  D0,(A1)      ;Output data
         JSR     Delay
         SUB     #1,D1        ;decrement loop count
         BNE     LOOP1        ;repeat for all 128 data
  :
Delay    MOVE    #deloop, D2  ;set up delay loop time
Loop2    SUB     #1,D2        ;decrement loop time
         BNE     Loop2        ;wait for loop time
         RTS                  ;return from subroutine

         ORG     $002000
Table    DS.B    128          ;128bytes reserved for data table
```

PSEUDO-PROGRAMME:

```
FOR i=1 to 128
    move data from
        table to port
    wait a fixed time
END FOR
```

Disadvantages:

Need delay time between outputs to be sufficient for external devices.

No handshake used

Microprocessor tied up by programme while waiting

Output to Port with polling (without interrupts)

```
Portdata    EQU       $08000     Location of Port data
Portstat    EQU       $08002     Location of Port's status byte
Count       EQU       128        Size of block to input

            ORG       $000400 Program origin
  :
            MOVE      #Count, D1    ;set up loop counter
            LEA       Table,A0     ;A0 points to table in memory
            LEA       Portdat,A1    ;A1 points to Port data
            LEA       Portstat,A2  ;A2 points to Port status
  :
  :
LOOP        MOVE.B    (A0)+,D0      ;D0←memory([A0])
                                    ; [A0]←[A0]+1
WAIT        MOVE.B    (A2),D2       ;Read status
            AND.B     #1,D2         ;mask off all but ready bit
            BEQ       WAIT          ;wait for port ready
            MOVE.B    D0,(A1)       ;Output data to peripheral
            SUB       #1,D1         ;decrement loop count
            BNE       LOOP1         ;repeat for all 128 data
  :
  :

            ORG       $002000
Table       DS.B      128           ;128bytes reserved for data table
```

PSEUDO-PROGRAMME:

```
FOR i=1 to 128
    get data from table
    wait until port ready
    output data
END FOR
```

Disadvantages:

Limited handshake

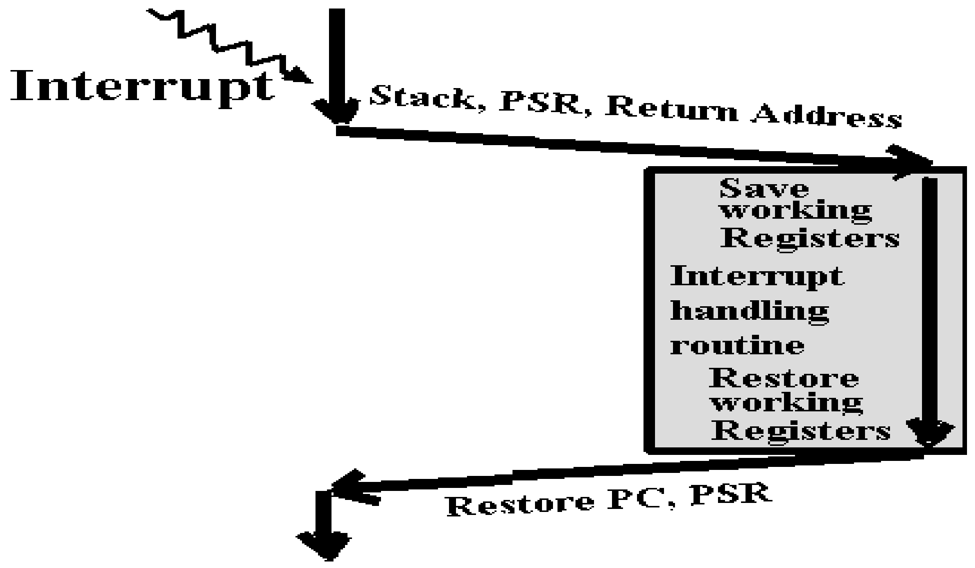
Microprocessor tied up waiting for peripheral to be ready

Need Interrupts...

Interrupt Driven I/O

Each interrupt vector to subroutine which:

Gets pointer for next entry, Reads a byte, Outputs to port, Moves pointer to next entry, Saves pointer in memory, Returns from interrupt


Interrupt
Stack, PSR, Return Address
Save working Registers
Interrupt handling routine
Restore working Registers
Restore PC, PSR

```
OUTPUT   EQU     $008000             Location of O/P Port
         ORG     $000400             Start of programme
  :
INT Y    MOVEM.L D0-D7/A0-A6,-(A7)             Save environment – general for subroutines
         MOVEA.L  POINTER,A0                   Point A0 to buffer
         MOVE.B (A0)+,D0                       Read a byte from buffer
         MOVE.B D0,Output                      Send to O/P port
         MOVE.L    A0,POINTER                  Save updated pointer
         MOVEM.L (A7)+,Do-D7/A0-A6             Restore Environment
         RTE                                   Return from interrupt
  :
         ORG     $002000            Data Origin
BUFFER  DS.B     1024                 Reserve 1024 bytes
POINTER DC.L     BUFFER               Reserve long word
```

In previous example (of last 2 slides): to obtain regular slow timed outputs- interrupt could be caused by a software pre-programmed Timer/counter chip output connected to a processor interrupt line.

Interrupt: Priority & Vectors

Interrupt Priority.

Example of 68000 has 7 levels of interrupt priority:

3 input pins IPL0-IPL2 can have values 0-7 (values negative logic)

0=no interrupt, 1=lowest priority interrupt → 7=highest priority level interrupt.

All interrupts at level ≥ 3bit mask in 68000 status word are serviced

Level 7 is thus a non-maskable interrupt - always serviced

Software can control when to service Interrupts < level 7

e.g. don't interrupt time critical processes

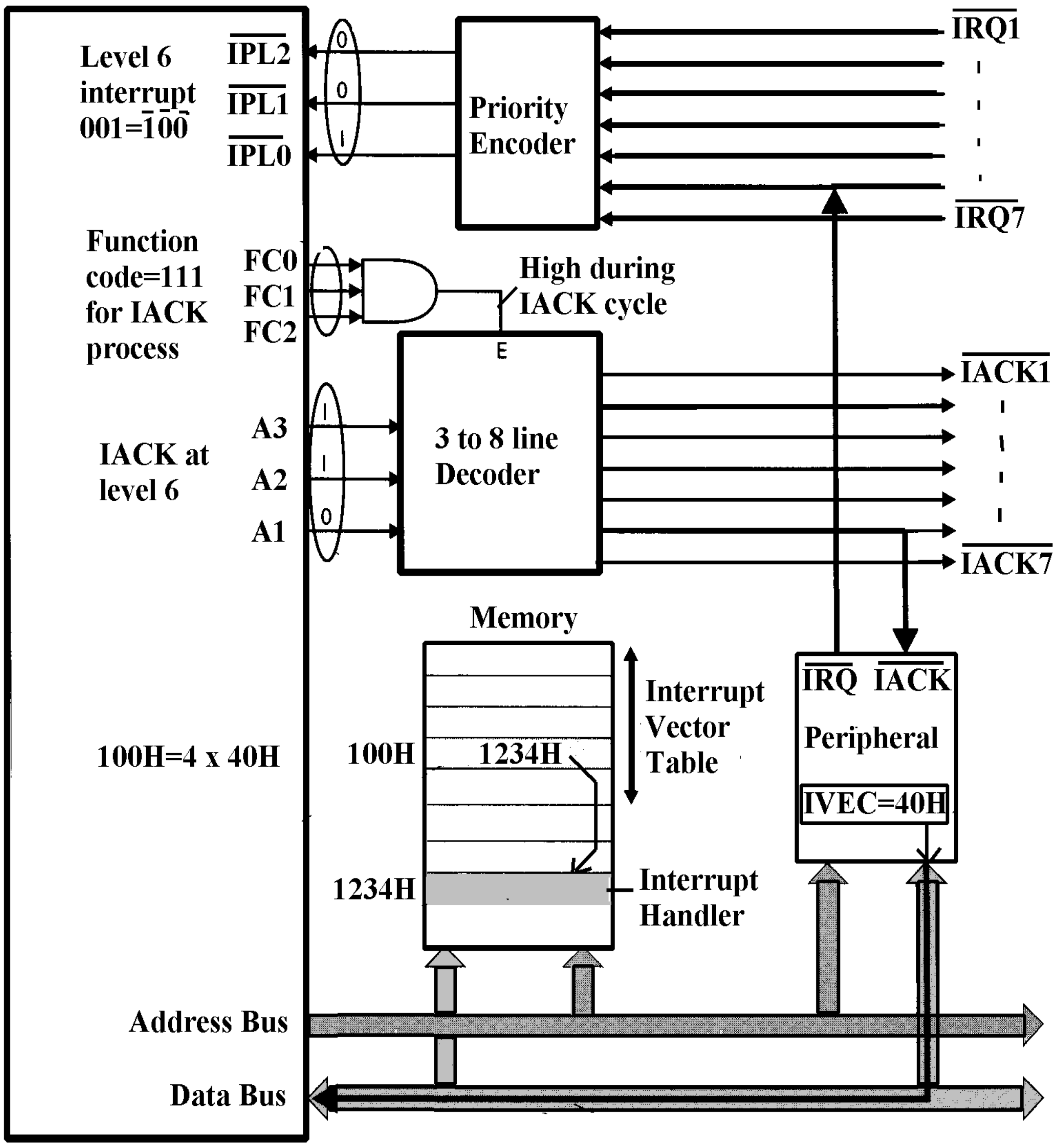
Interrupt Address Vectors (Interrupt programme control sequence)

Peripheral provides interrupt signal to Processor

Processor acknowledges to peripheral that it will accept interrupt

Peripheral provides interrupt vector to processor

Processor uses vector to look up location of interrupt handler routine



Multi – Peripheral Interrupt + Acknowledge

Each Peripheral:
provides interrupt
receives acknowledge

Priority Encoder:
converts IRQ1-IRQ7 to
three bits IPL0-IPL2

IACK Decoder:
decodes CPU response
function code = IACK
+ address = level
& generates IACK1-6

Peripheral with IACK:
Provides interrupt vector
e.g. 40H

CPU:
gets interrupt vector from
memory pointed to by
peripheral vector x 4 e.g =100H

Part 5.
Serial I/O:

- Asynchronous and synchronous transmission

- UARTs

- Serial I/O under program control

- Other standards

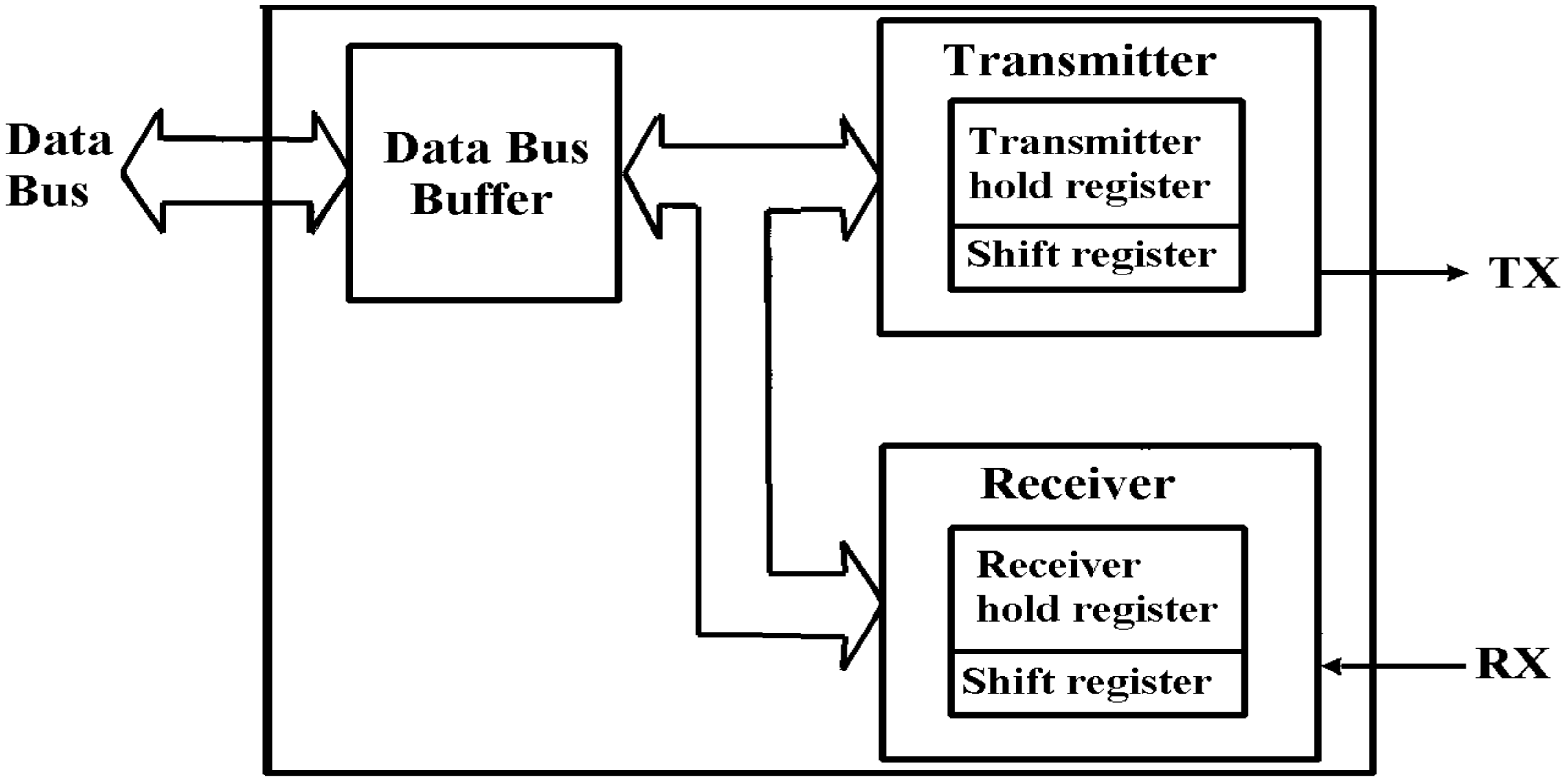
Serial Interfaces

Serial Transmission can be:

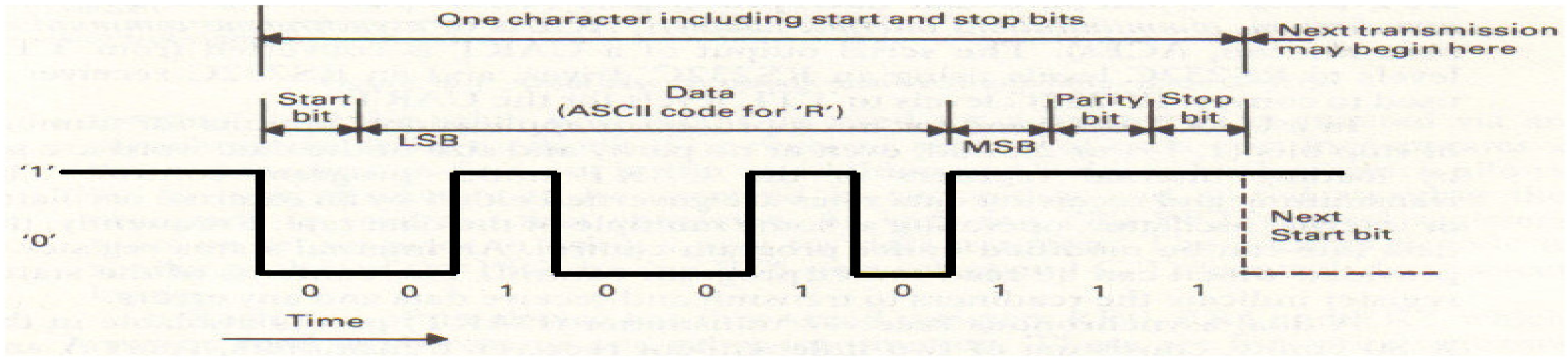
i) Asynchronous (e.g. traditional PC COM1 port)

ii) Synchronous (e.g. USB Port)

UART chip performs parallel-to-serial conversion on data sent from CPU and serial-to-parallel conversion on data received by CPU. Mechanism of shift register, shift out bits of data byte (or character) one at a time.



Bit-Serial Data


One character including start and stop bits
Next transmission may begin here
Start bit
Data (ASCII code for 'R')
Parity bit
Stop bit
LSB
MSB
'1'
'0'
Next Start bit
0 0 1 0 0 1 0 1 1 1
Time

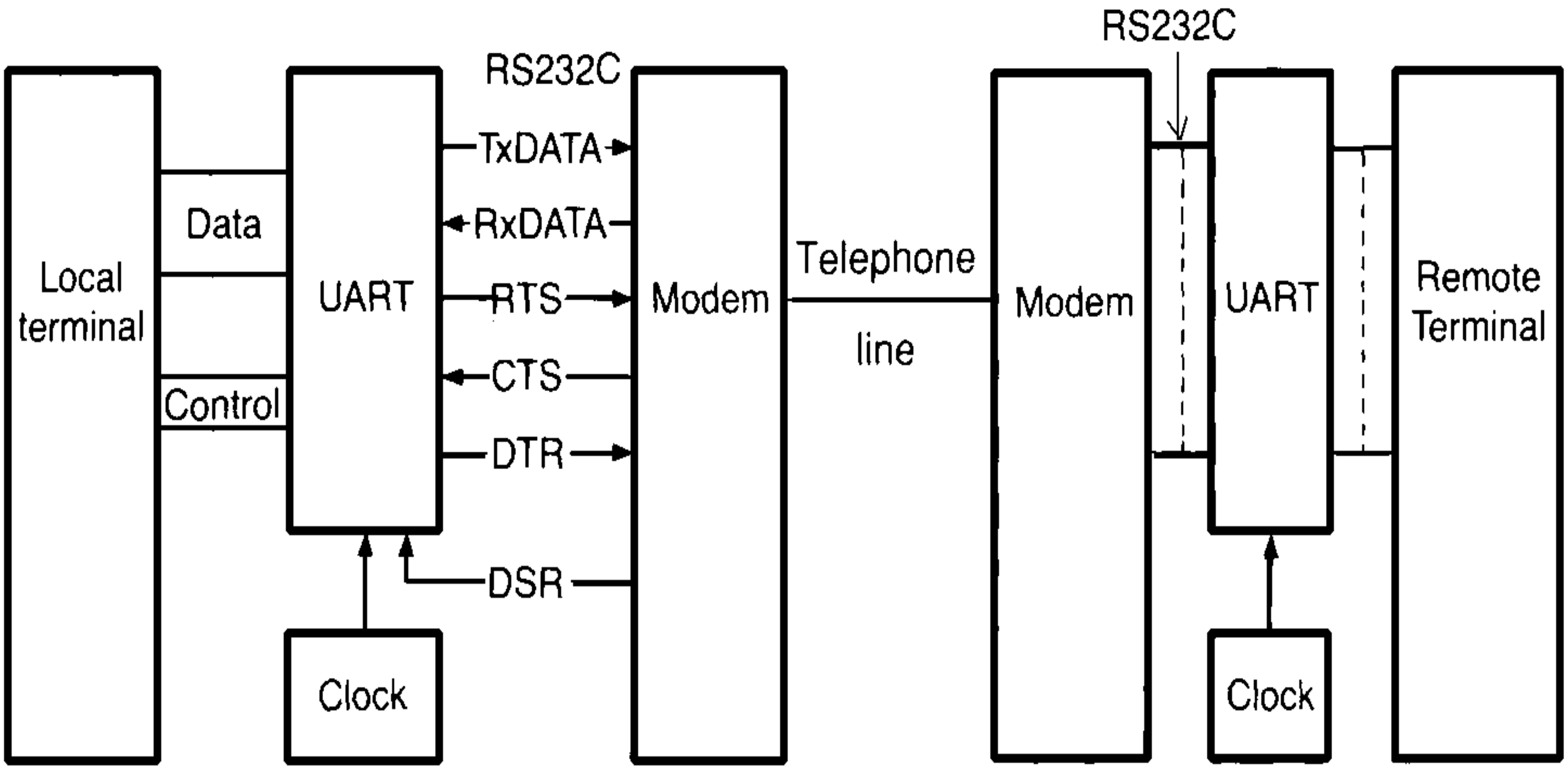
Non-Return to Zero (NRZ), quiescent level='1'
Bit serial data framed by start and stop bits with optional parity bits for error checking. E.g. if 7bit character data(ASCII) then up to 11 bits required per character.
Above example of transmitting character 'R',
in ASCII is 52Hex (1010010b). Seven bit data. (8th MSBit discarded).

Character rate = bit rate / bits per character **Bit rate = "baud rate"**

Data Link can be:

1) **Simplex (one way data transfer)**
2) **Half Duplex (two way data transfer, but only one at a time)**
3) **(Full) Duplex (two way data transfer simultaneously)**

Typical example of serial communications



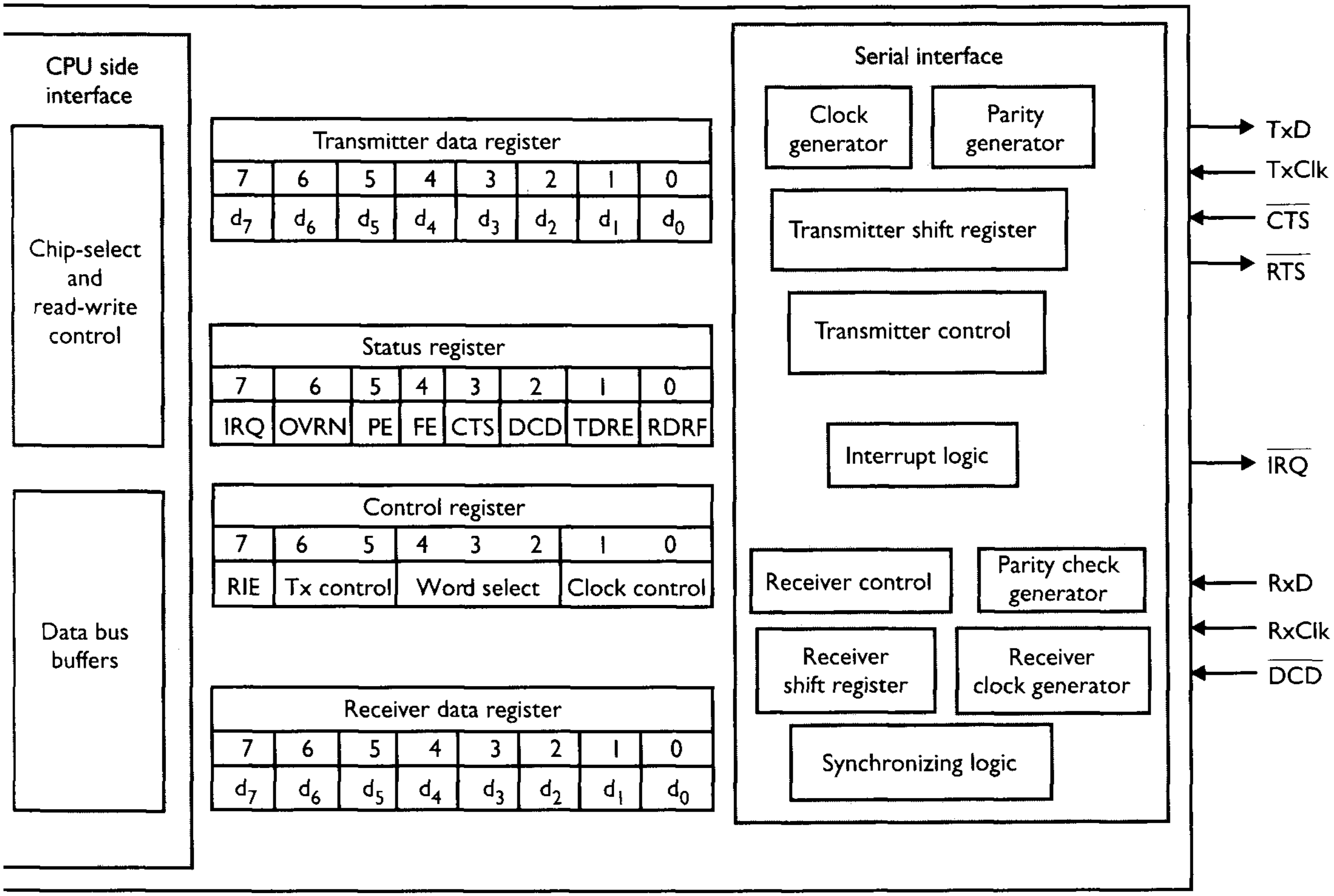
Extra signals needed for handshake with external serial devices:

RTS: Request to send. Computer asks modem if it is ready for data operations

CTS: Clear to send. In response to RTS modem tells computer data can be sent

DCD: Data carrier detect. Modem tells computer that it receives carrier tone on the telephone line

UART for 68000 - Asynchronous Communications Interface Adapter)



Using the ACIA: Example Software

Configuring ACIA:

```
ACIA     EQU      $800000               ;ACIA address
CR       EQU      0                     ;Control Register Offset
         LEA      ACIA,A0               ;A0 points to CR
    MOVE.B #%00000011,CR(A0)            ;software reset   (user looks up codes in datasheet)
    MOVE.B #%10110101,CR(A0)            ;set baud rate, handshake, interrupt (  " " )
```

Receive a Character Subroutine:

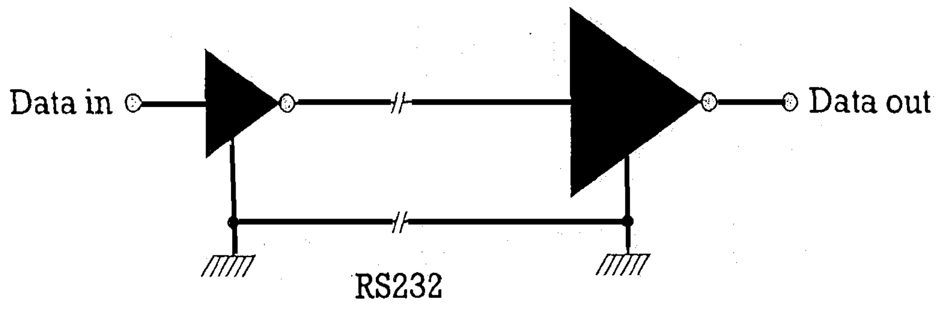
```
RDRF     EQU      0                 ;RX data ready bit 0 of SR
SR       EQU      0                 ;Status register offset
DR       EQU      2                 ;Data register offset
         LEA      ACIA,A0           ;A0 points to ACIA
POLL     TST.B    #RDRF,SR(A0)      ;Read RX status bit
         BEQ POLL                   ;repeat until char received
         MOVE.B   DR(A0),D0         ;get input from ACIA to D0
         RTS
```

Transmit a Character Subroutine:

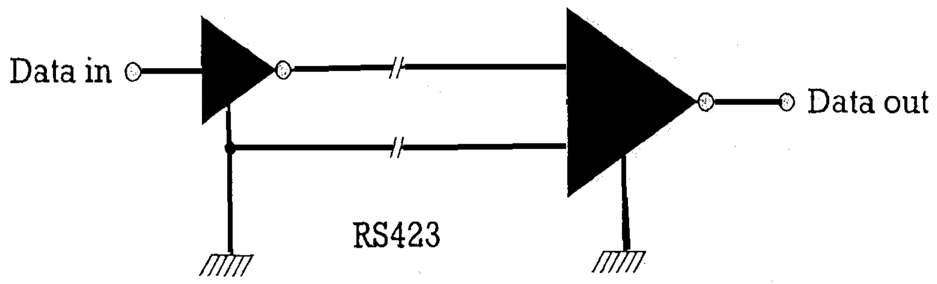
```
TDRE     EQU      1                 ;Transmitter data register empty bit
         LEA      ACIA,A0           ;A0 points to ACIA
TPOLL    BTST.B #TDRE,SR(A0)        ;TX register empty?
         BEQ TPOLL                  ;Repeat until ready to transmit
         MOVE.B D0,DR(A0)           ;Move byte from D0 to ACIA
         RTS
```

Types Of Serial Interface:

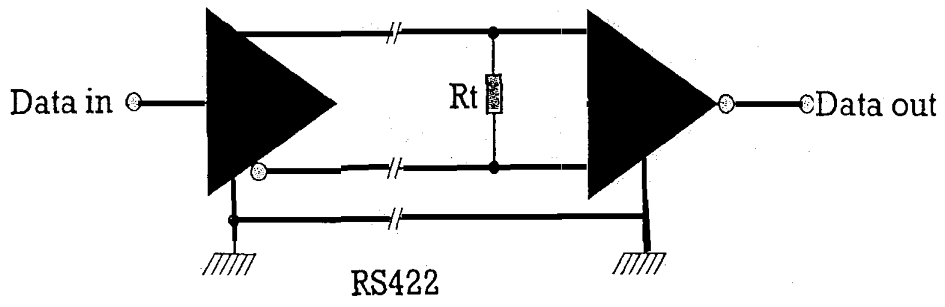
RS232C: (Bipolar) Logic 1 has value < -3V (typically -12V)
Logic 0 has value > +3V (typically +5V).



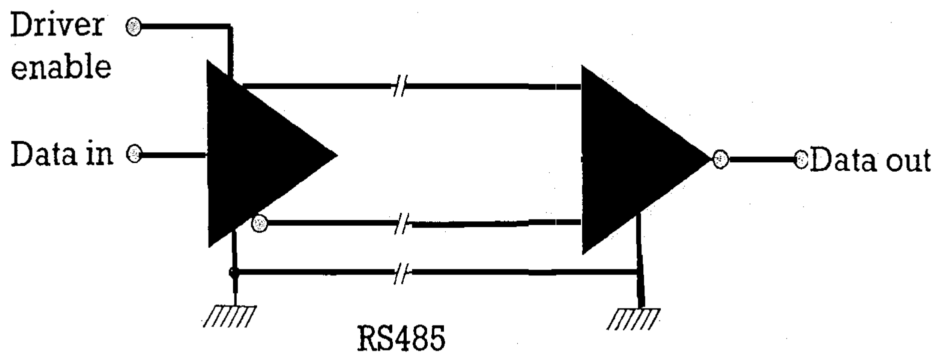
RS423: Low impedance 50Ω



RS422: Low impedance differential twisted pair eliminates ground loop pickup,etc

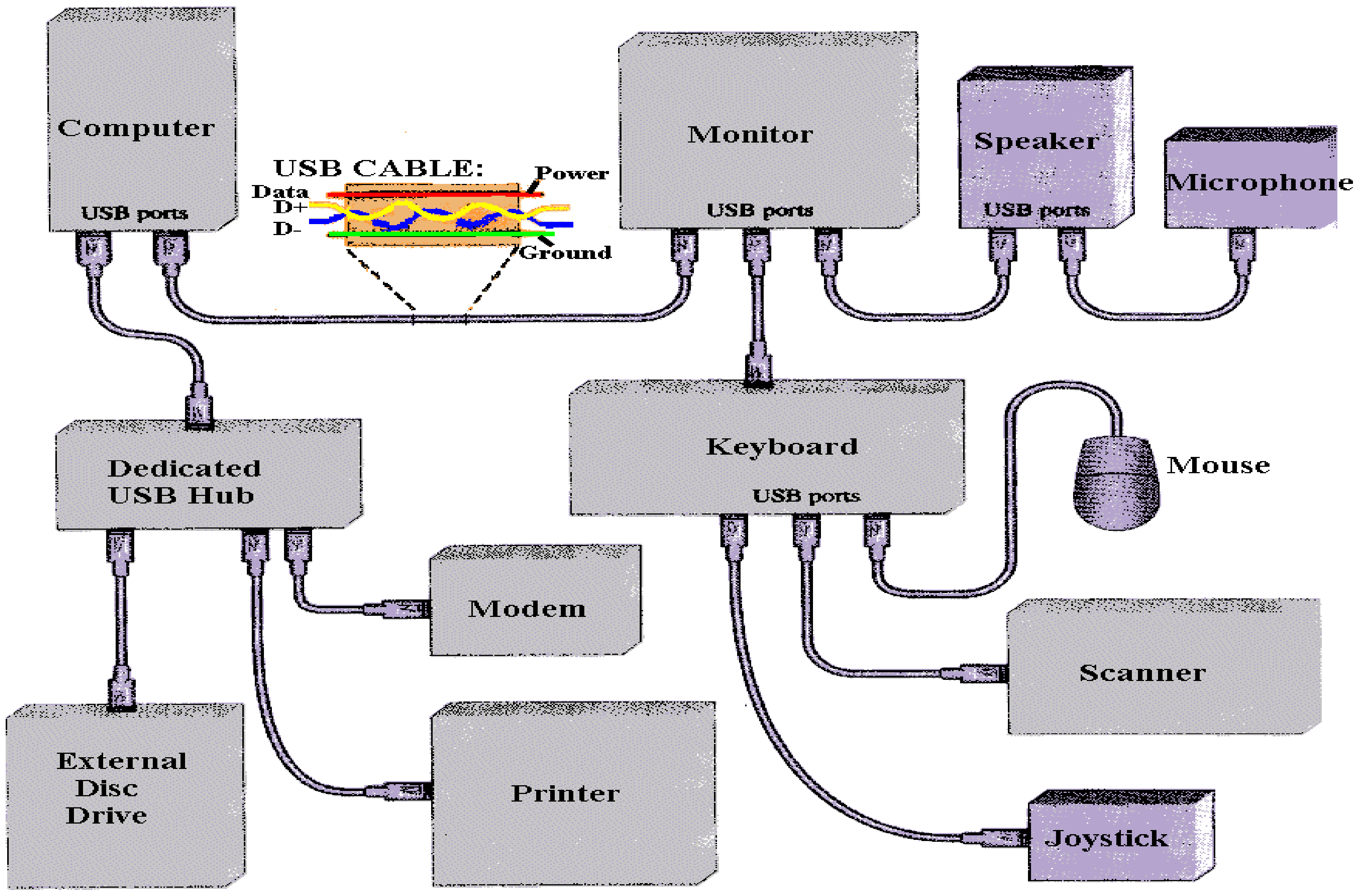


RS485:



Universal Serial Bus, USB

fast data + can power small devices from bus e.g. flash memory



Some other Standards:

Firewire (IEEE 1394)

External serial bus for fast data transfer 400Mb/s (developed by Apple).
Up to 63 devices can be connected in daisy-chain arrangement.
6 wires: two twisted pair for data ←→, power, ground

GPIB, General Purpose Interface Bus (IEEE488).

Parallel Bus developed by Hewlett-Packard for test & measurement equipment 1MByte/s.
24 lines: 8 data lines, 8 ground returns/screening, 3 handshake lines, 5 bus-management lines.

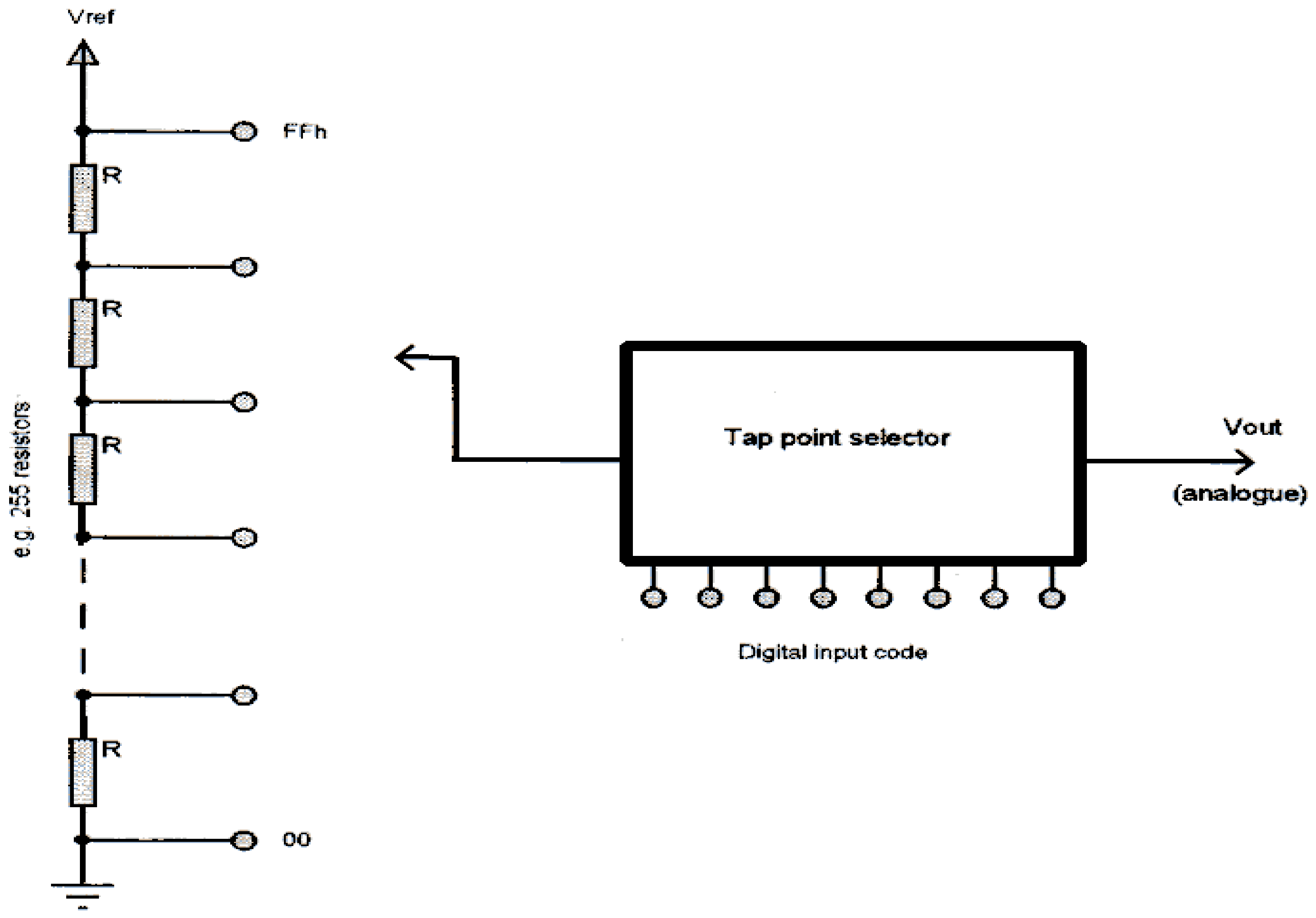
SCSI, Small Computer System Interface (“scuzzy”).

Parallel Bus with various variations & connectors
e.g.
a) SCSI-1- 25 pin connector 8-bit data + handshake, upto 4Mbytes/s
b) Wide Ultra SCSI-2, 16bit data at 80Mbytes/s

Part 6.
Analogue I/O (or Digital meets the real World):

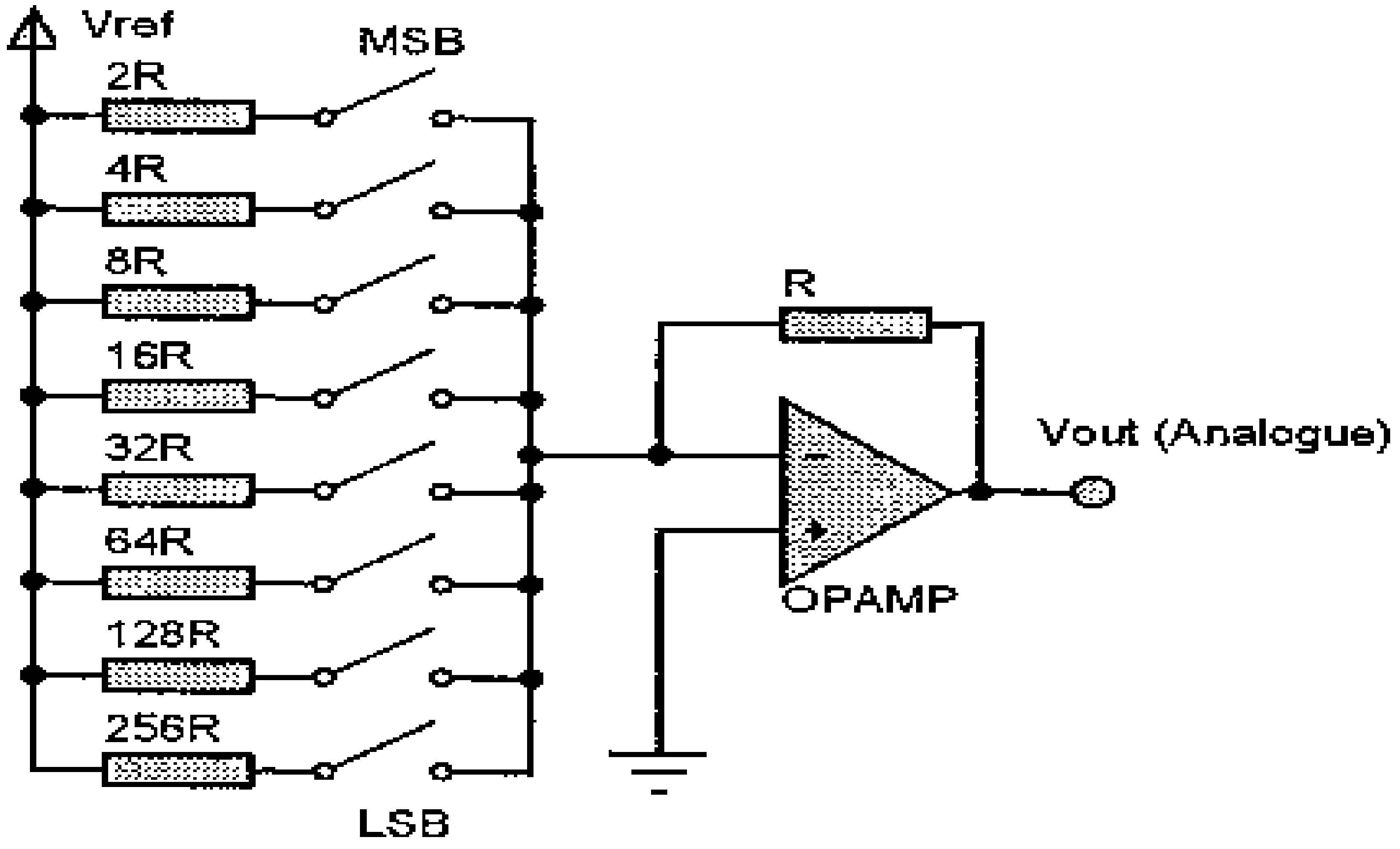
- Digital-to-Analogue (DAC) principles
- Analogue-to-Digital (ADC) principles
- Software Interfacing methods
- Sampling and aliasing
- Programming techniques
- Introduction to digital filtering

Potential Divider Network DAC



Many resistors needed - 2^n where n= number of bits, but all same value

D to A: Type= Binary-Weighted-Input DAC



a) Simple Explanation: Each bit if logic ‘1’ connects resistor to the circuit. R values vary as 2:1 from bit to bit with MSB having the lowest value R low R →passes highest current → most effect on output voltage.

Disadvantage: need to have many different, precise R values

b) Detailed Explanation: Amplifier -ve input is virtual ground since feedback resistor from V_{out} holds inputs at 0 volts. High impedance input amplifier takes zero input current, so all currents I_0, I_1, etc, must pass through feedback resistor, R_f.

Total current in feedback resistor, $I_f = b_0 I_0 + b_1 I_1 + b_2 I_2 + \ldots$ (With each bit b_n =0 or =1)

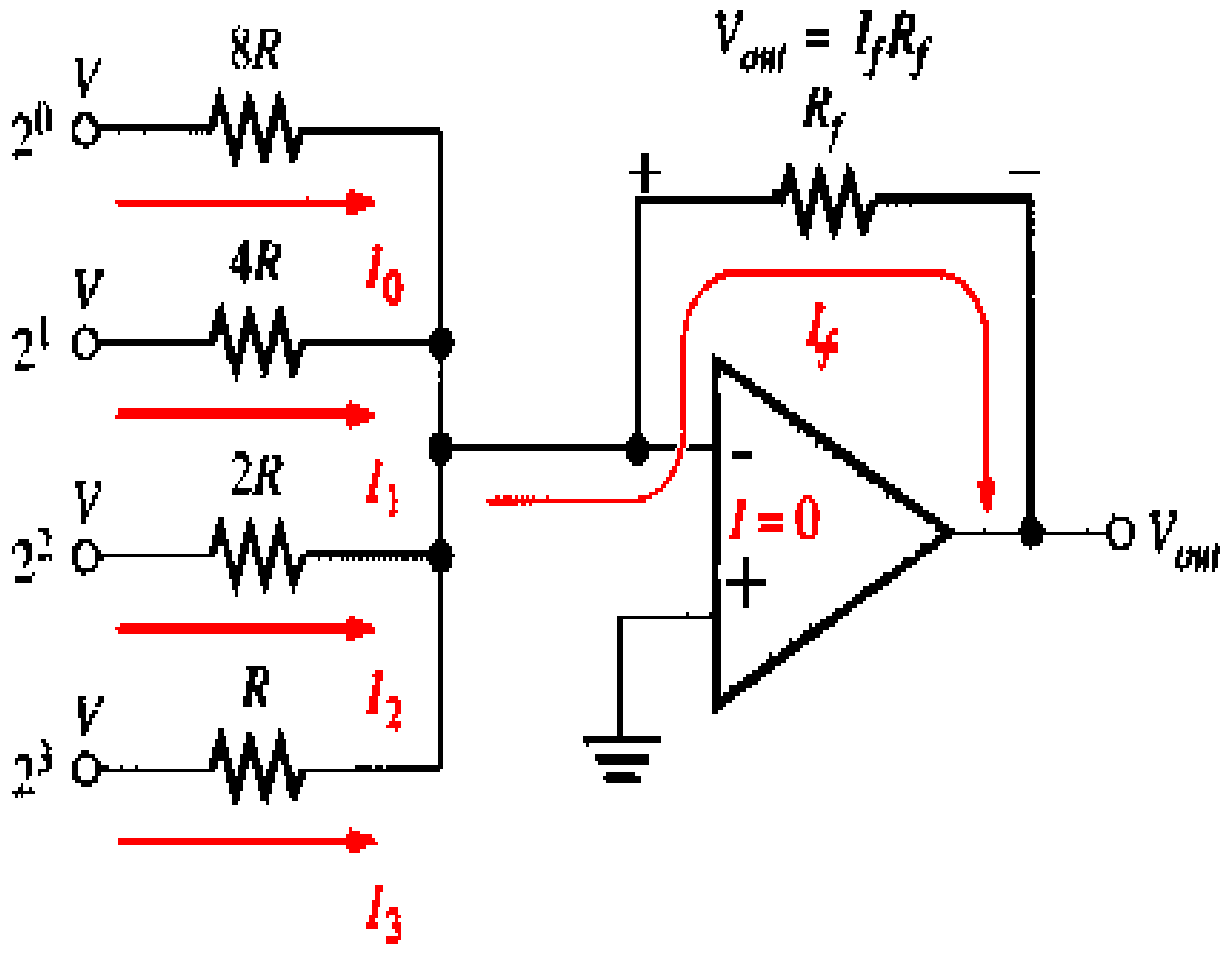
So final analogue voltage output= $V_{out} = I_f R_f$

$$I_0 = \frac{V}{8R}$$

$$I_1 = \frac{V}{4R}$$

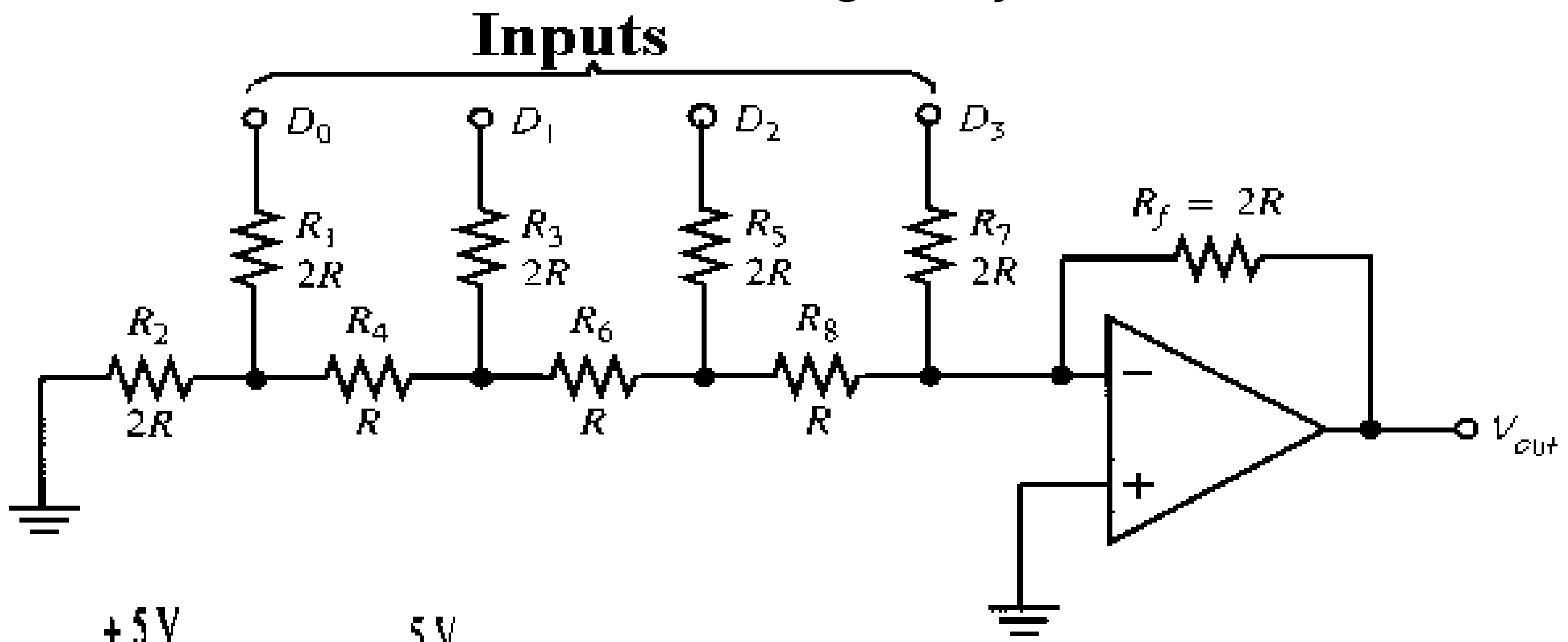
$$I_2 = \frac{V}{2R}$$

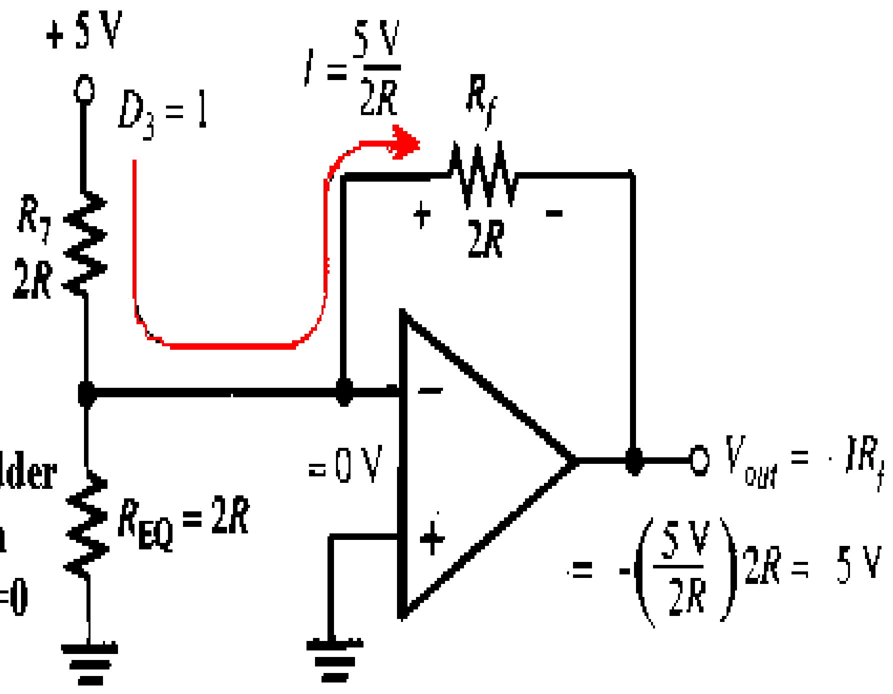
$$I_3 = \frac{V}{R}$$



An R-2R Ladder type DAC

Advantage: only two different R values: R & 2R




+5 V
$D_3 = 1$
$I = \frac{5\text{ V}}{2R}$
R_f
$2R$
R_7
$2R$
≈0 V
$R_{EQ} = 2R$
$V_{out} = -IR_f$
$= -\left(\frac{5\text{ V}}{2R}\right)2R = -5\text{ V}$

Equivalent ladder resistance with D0,D1,D2 all =0

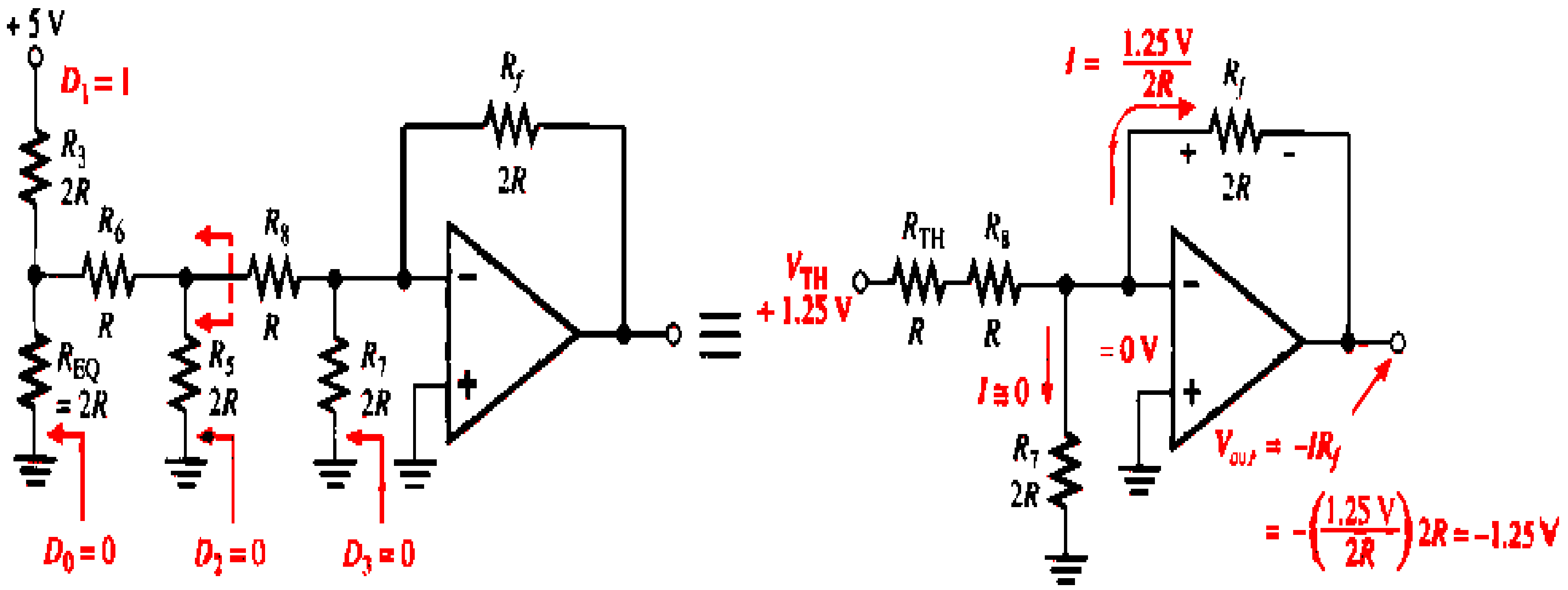
Equivalent circuit for input, D3,D2,D1,D0=1000

Example: data value 1000

D3=1 →5V via 2R to input held at 0V by feedback- all current flows through Rf(=2R) so V_{out} must be -5V.

Lumped value, Req of other resistors not critical as no current passes through Req

R-2R ladder DAC- another example value = 0010



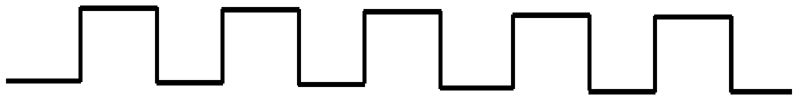
Equivalent circuit for $D_3 = 0, D_2 = 0, D_1 = 1, D_0 = 0$

Thevenin's Theorem- any circuit can be reduced to an equivalent voltage in series with an equivalent resistor.
Applying theorem to the left of R8 we have Vth=1.25V & Rth=R.
Again voltage across R7=0, then 1.25V through 2R to input will require Vout to be -1.25V to keep input at zero voltage (Virtual earth).

Waveform Generator Pseudo-Programmes

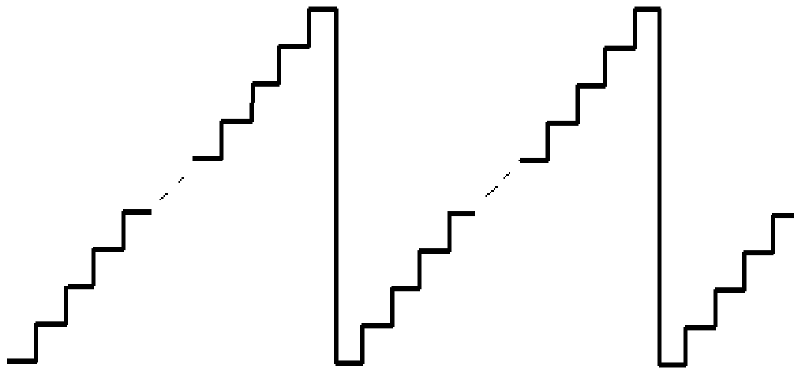
a) Square wave period T (without DAC):

```
Loop1  Output '0' on port pin
       wait T/2
       Output '1' on port pin
       wait T/2
       branch to Loop1     repeat forever
```



b) Sawtooth ramp period T (with 8 bit DAC)

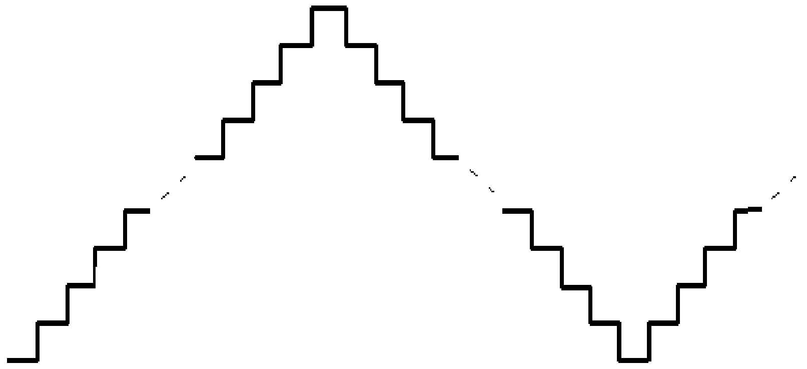
```
       initialise D0=0
Loop2  output D0 to DAC
       increment D0
       wait T/256
       branch to Loop2    repeat forever
```



256 steps up/down

c) Triangular wave period T (with 8 bit DAC)

```
       initialise D0=0
Loop3  output D0 to DAC
       increment D0
       wait T/512
       compare D0 to #255
       branch to Loop3 if not equal
Loop4  output D0 to DAC
       decrement D0
       wait T/512
       compare D0 to #0
       branch to Loop4 if not equal
       branch to Loop3   repeat forever
```



DAC performance aspects:

Resolution

Improves with number of bits, n.   % resolution = 100 * 1 / $(2^n – 1)$

Accuracy

Ideally = resolution but in practice less because of accuracy of resistors

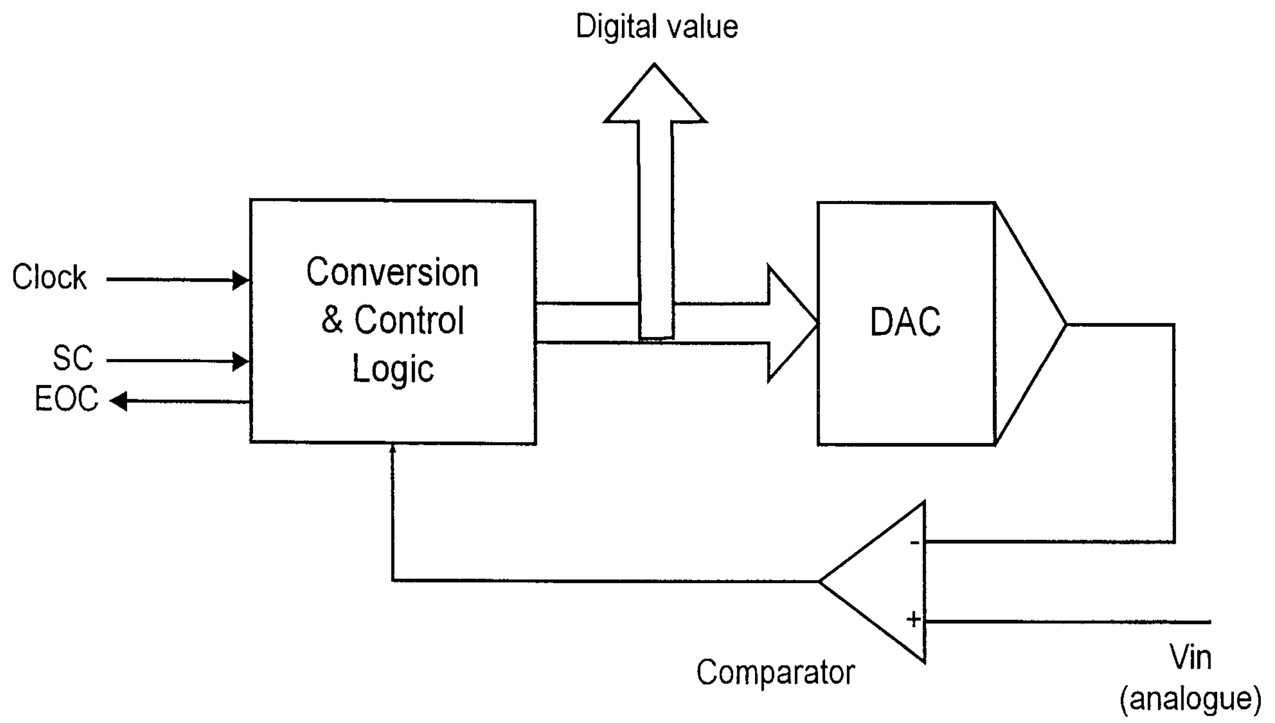
Linearity

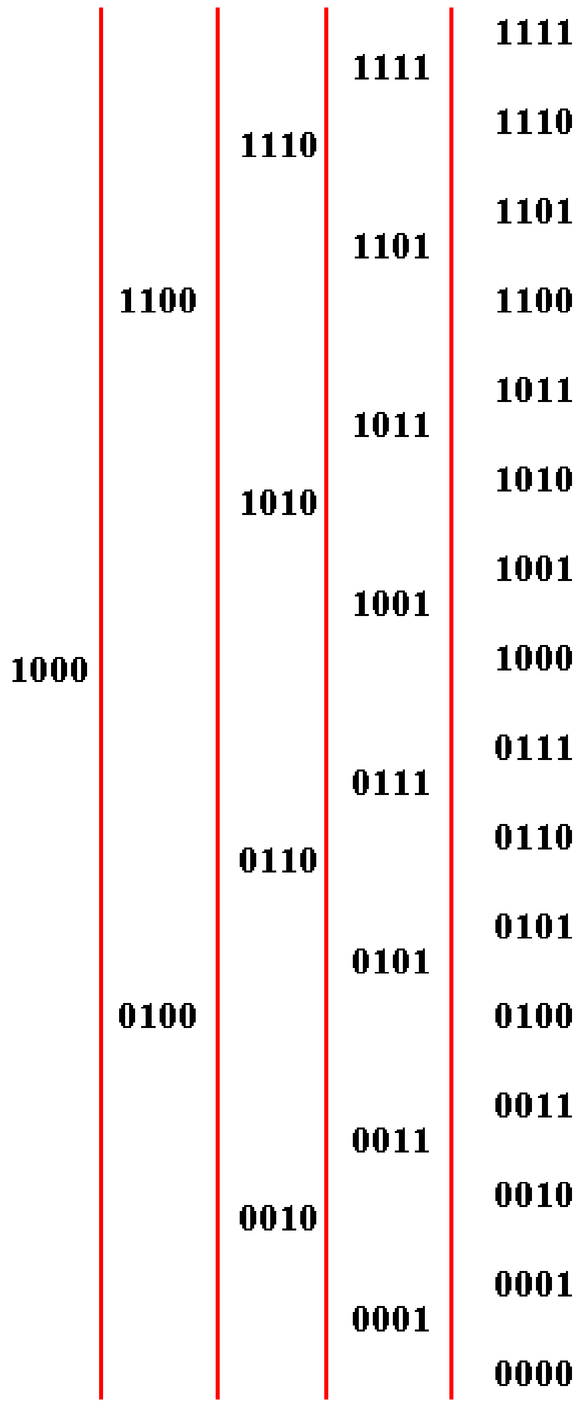
Linear error is deviation from ideal straight line V_{out}= constant x digital value

Settling Time

Time taken for analogue output value to reach a new value in response to a change in the digital input - depends on RC time constants , internal & external capacitance.

Analogue to Digital Conversion: Successive-Approximation





Example of 4bit ADC

1) Microprocessor sends SC (start conversion) to control

2) Control logic within ADC outputs a digital value to a DAC

3) The analogue input is compared with the DAC output

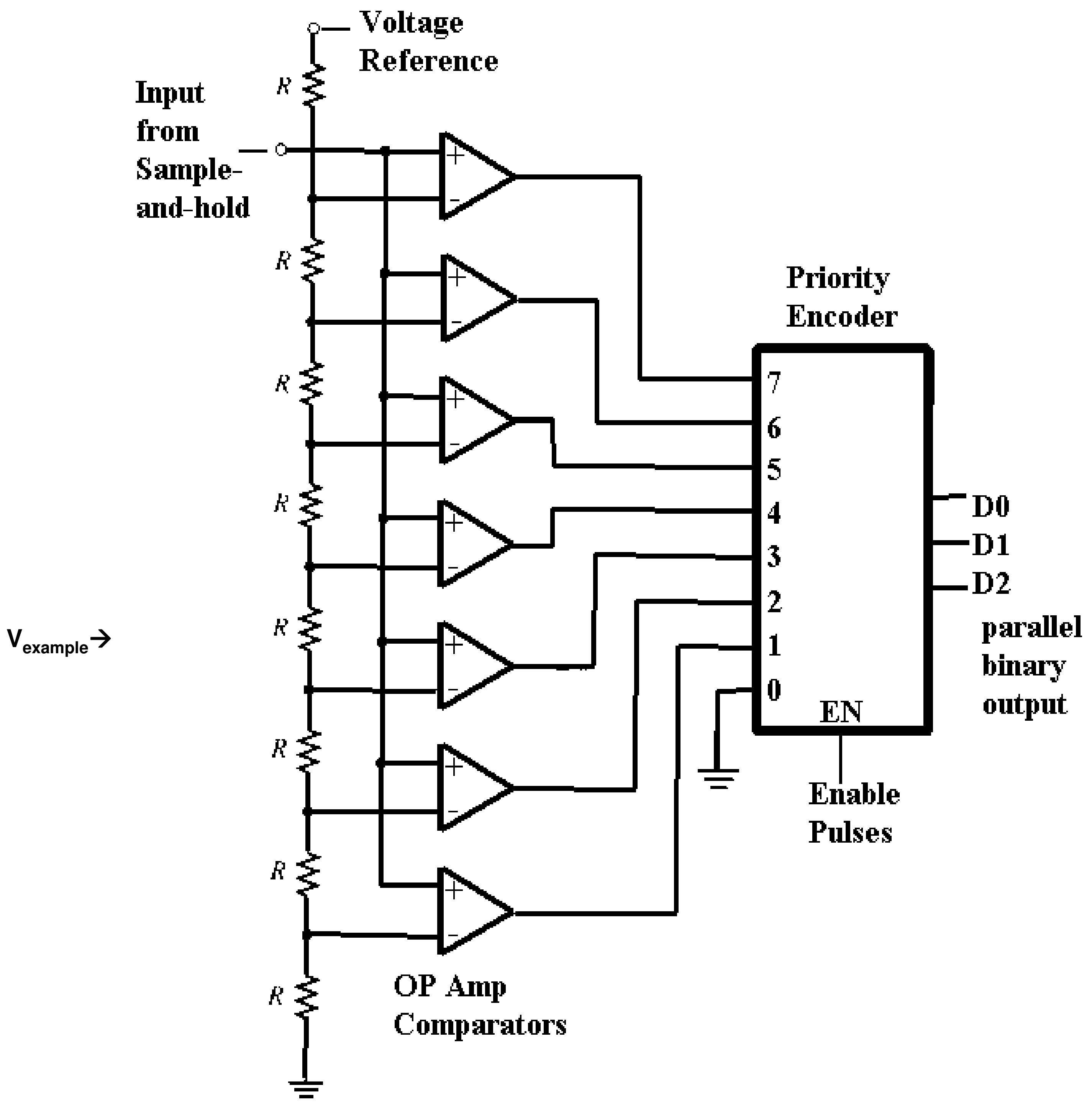
3) Control logic tries each bit in turn starting at MSB

Decision tree: Only four decisions (red lines) →

4) After 4 successive approximations sends end of conversion EOC signal to microprocessor

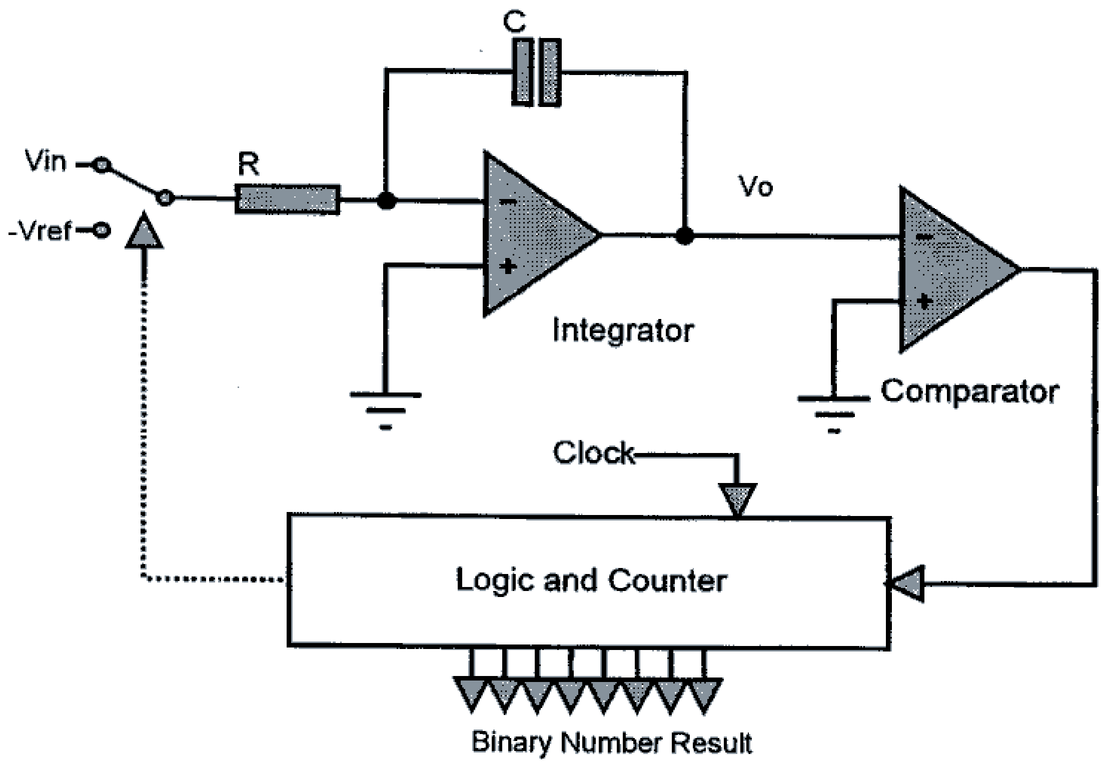
5) Microprocessor reads digital value

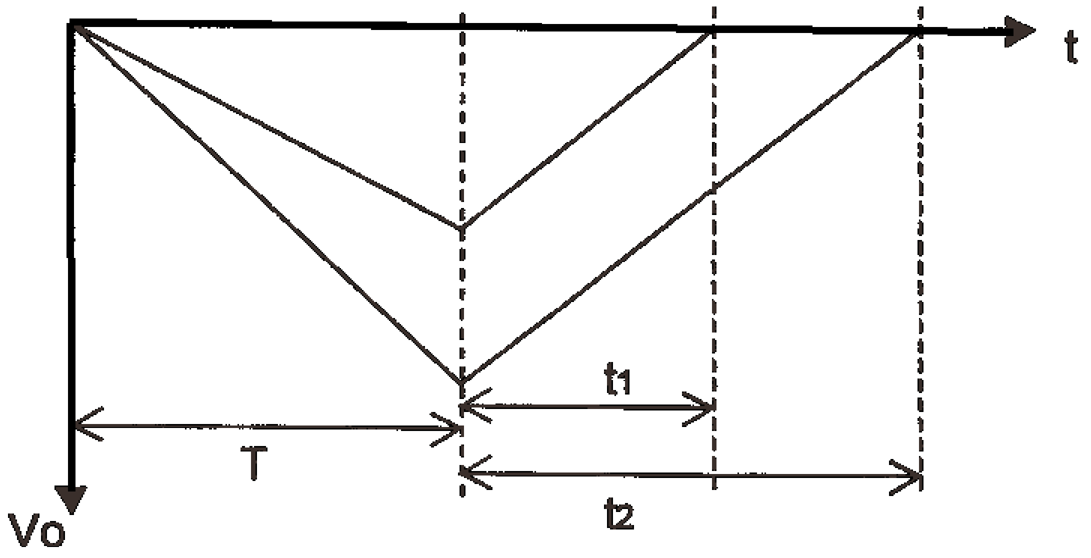
Flash ADC
3-bit example



Input analogue signal fed in parallel to many comparators which compare input against a voltage divider chain.
Bits set from bit 0 to the voltage tap just below the input voltage value.

An encoder then converts the signal to binary value.
e.g. $V_{in} = V_{example}$
7… … .1
0000111 → 011_2





Dual Slope ADC

Start: assume counter is zero and output of integrator=0.

1) Switch connects +ve V_{in} to R assume V_{in} steady (sample/hold)

2) C charges linearly(const I=V/R) resulting in negative ramp at V_0

3) When counter reaches a preset value (time T) counter reset, & control switches input to $-V_{ref}$ causing C to discharge linearly towards zero.

4) When Vo reaches zero comparator stops count.

5) Count value t1 or t2 depends on size of the input voltage.

6) Binary count t1/t2 read out

Main ADC Performance Aspects:

- Conversion Time

  (application specific & the need to avoid aliasing)

- Conversion Accuracy

  (increases with number of bits)

- Electrical Power consumption may be limited in small systems

  (power increases with conversion rate)

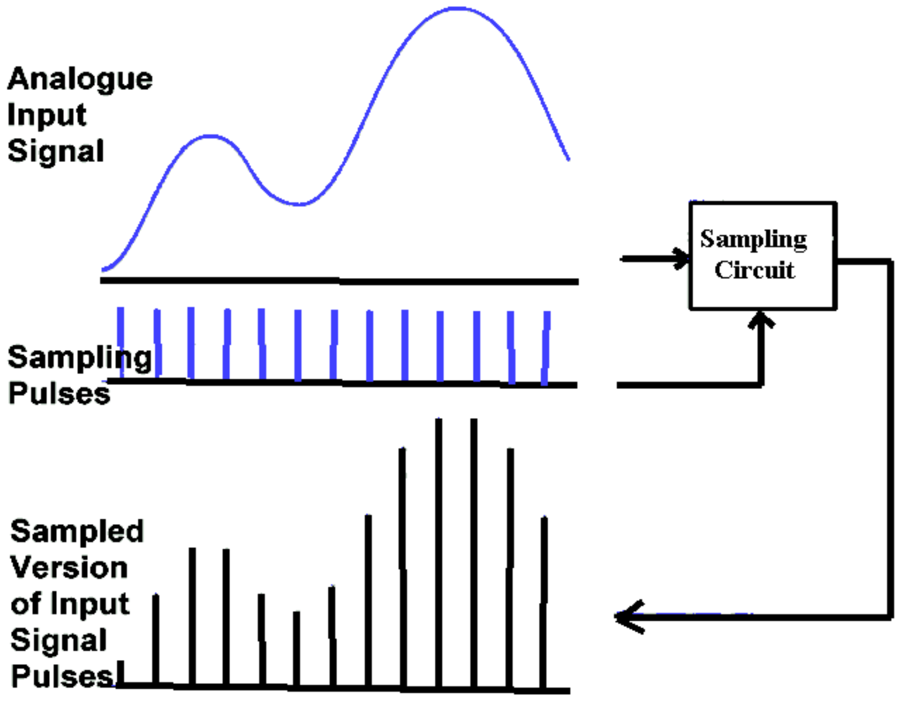
Example software to access ADC

```
ADCstatus        EQU $8001             ADC status register address
ADCdata          EQU $8000             ADC data register address
Size             EQU $80               Number of values to read into table
Table            EQU $4000             Address of destination table in memory

        MOVEA.W A0,ADCstatus           A0 points to ADC status register
        MOVEA.W A1,ADCdata             A1 points to ADC data register
        MOVE D1,$Size                  D1 holds the number of values to read
        MOVEA.W A2,table               A2 points toTable of values read from ADC

Loop    MOVE.B $01,(A0)                Start ADC Conversion- set SC bit
Wait    MOVE D0,(A0)                   Read ADC Status
        AND A,$01                      Mask off bits other than EOC bit
        BNZ Wait                       Wait for End of Conversion EOC
        MOV (A1),D0                    Read ADC value
        MOV D0,(A2)+                   Store in table, increment table position
        SUB $01, D1                    Decrement Loop counter
        BNZ Loop                       Repeat to complete data table
```

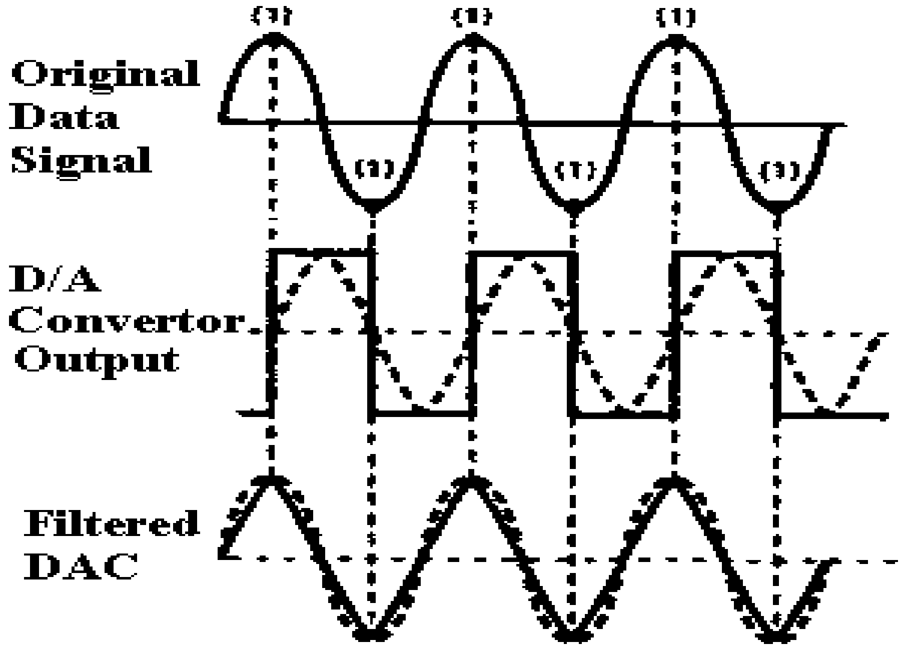
Sampling & Aliasing Error

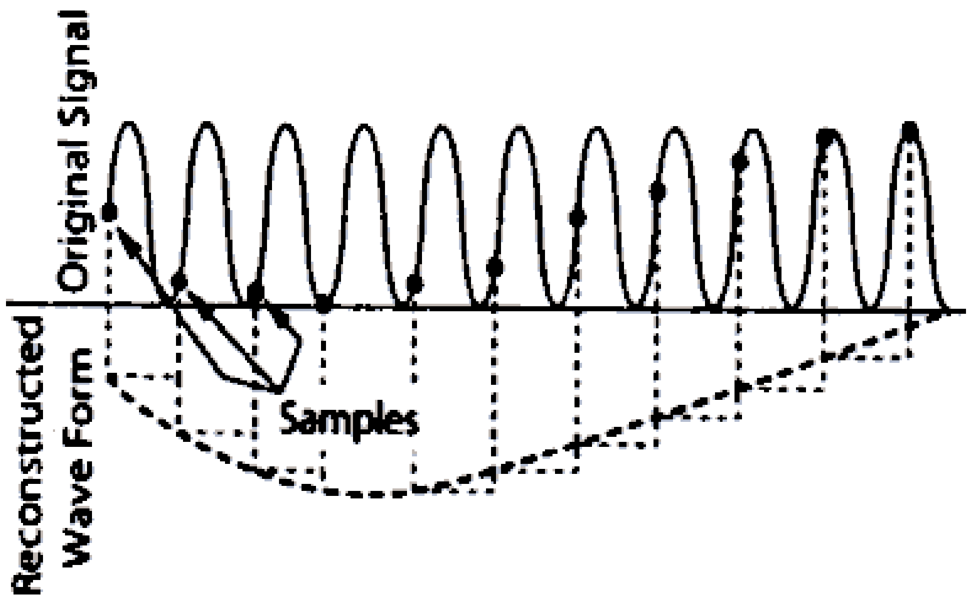


a) For good reconstruction of signals the sampling frequency, f_{sam}, should be $> 2 f_{max}$, where f_{max} is the maximum signal frequency or, $f_{max} \leq$ Nyquist frequency (= $f_{sam}/2$). Example of sufficient sampling: *figure on left*.

b) Absolute limit of 2 samples per wave cycle $f_{sam} = 2 f_{max}$ (*figure lower left*).

c) Aliasing errors occur when $f_{sam} < 2 f_{max}$ as illustrated here in the *figure below right.* i.e. less than 2 samples per wave cycle. The reconstructed waveform is then a very different frequency from the original signal.

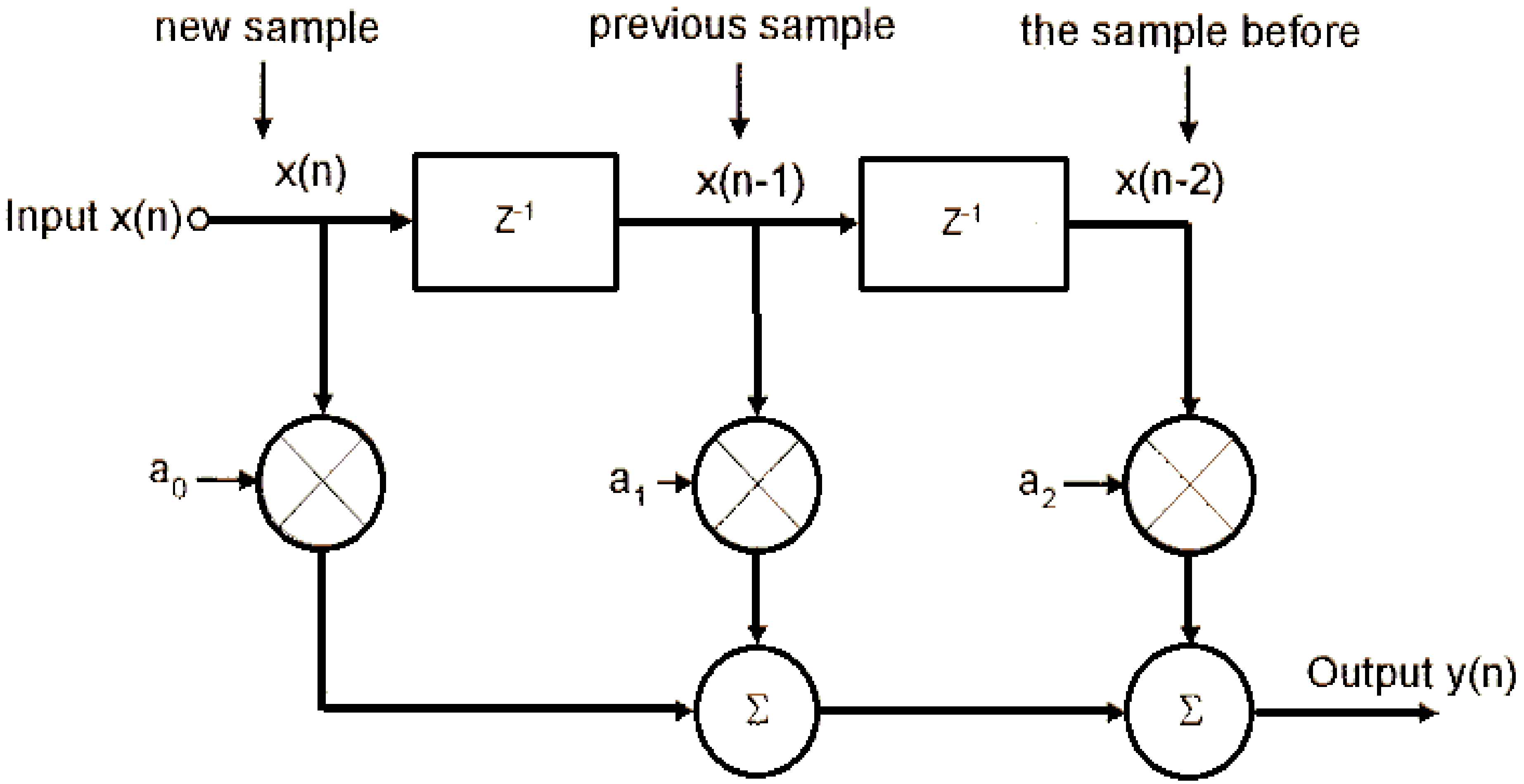




Simple Digital Processing Example: e.g. moving average filter

Analogue signal → ADC → Digital Processing → DAC → Processed Analogue signal

Typical Filter Processing:



x(n)=sampled analogue waveform,

a_n =weights (coefficients, or scaling factor),

Z^{-1} =unit time delay = 1 sample period

Simple Digital Filtering

Moving Average FIR Filter:

Specific ADC → input, x(n) → {processing} → result, y(n) → DAC

$$y(n) = (1/4)\{ x(n) + x(n-1) + x(n-2) + x(n-3)\}$$

Use four registers D0,D1,D2,D3 to store signal samples x(n), x(n-1), x(n-2), x(n-3)

LOOP

- Read new ADC value
- Store this new value in D0
- Add D0 to D1
- Divide by 2 (arith shift right) & store in D4 $= (1/2)\{ x(n) + x(n-1)\}$
- Add D2 to D3
- Divide by 2 & store in D5 $= (1/2)\{ x(n-2) + x(n-3)\}$
- Add D4 to D5
- Divide by 2
- Output this value to DAC $= (1/4)\{ x(n) + x(n-1) + x(n-2) + x(n-3)\}$
- Move D2 contents to D3 $x(n-2) \rightarrow x(n-3)$
- Move D1 contents to D2 $x(n-1) \rightarrow x(n-2)$
- Move D0 contents to D1 $x(n) \rightarrow x(n-1)$
- Repeat LOOP forever get new x(n)

Note: No need to initialise D1-D3 as not important after 3 programme loops.

Part 7.
Microcontrollers for small embedded systems:

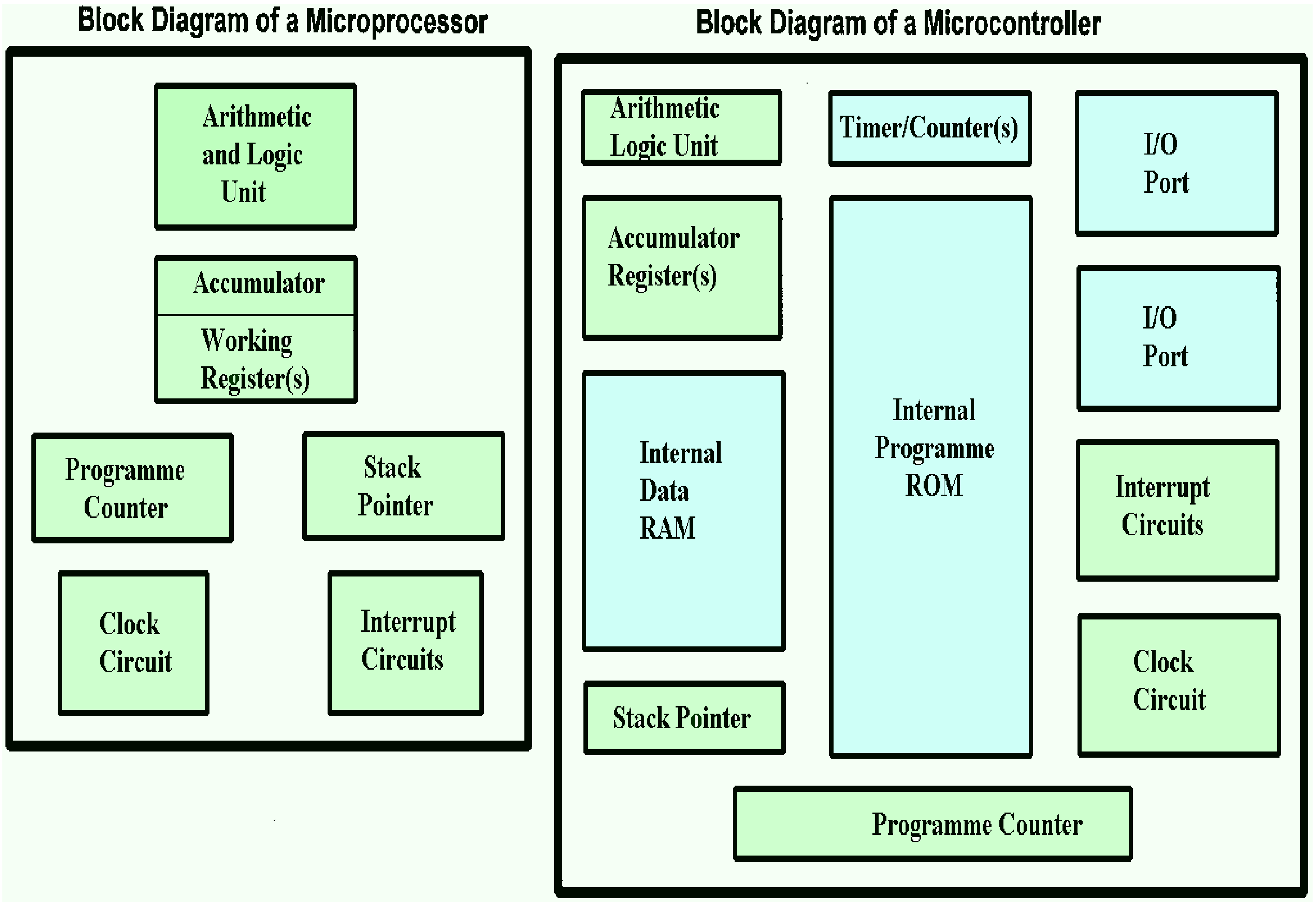
- Configurations

- Architectures

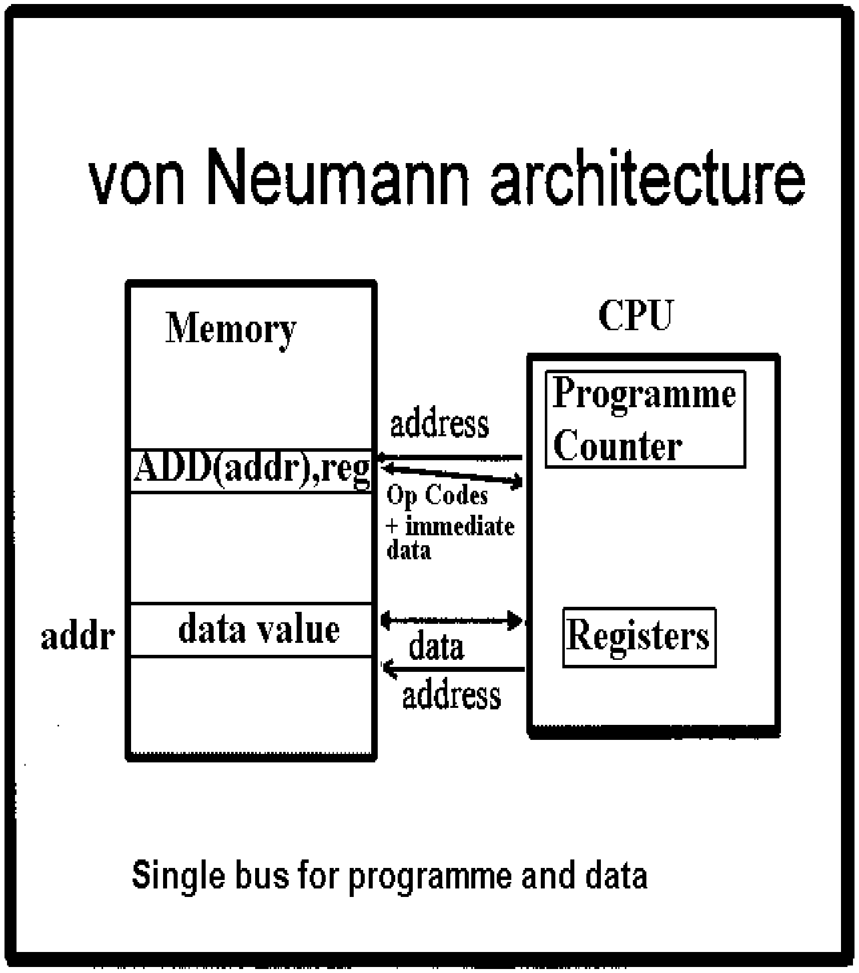
- Features

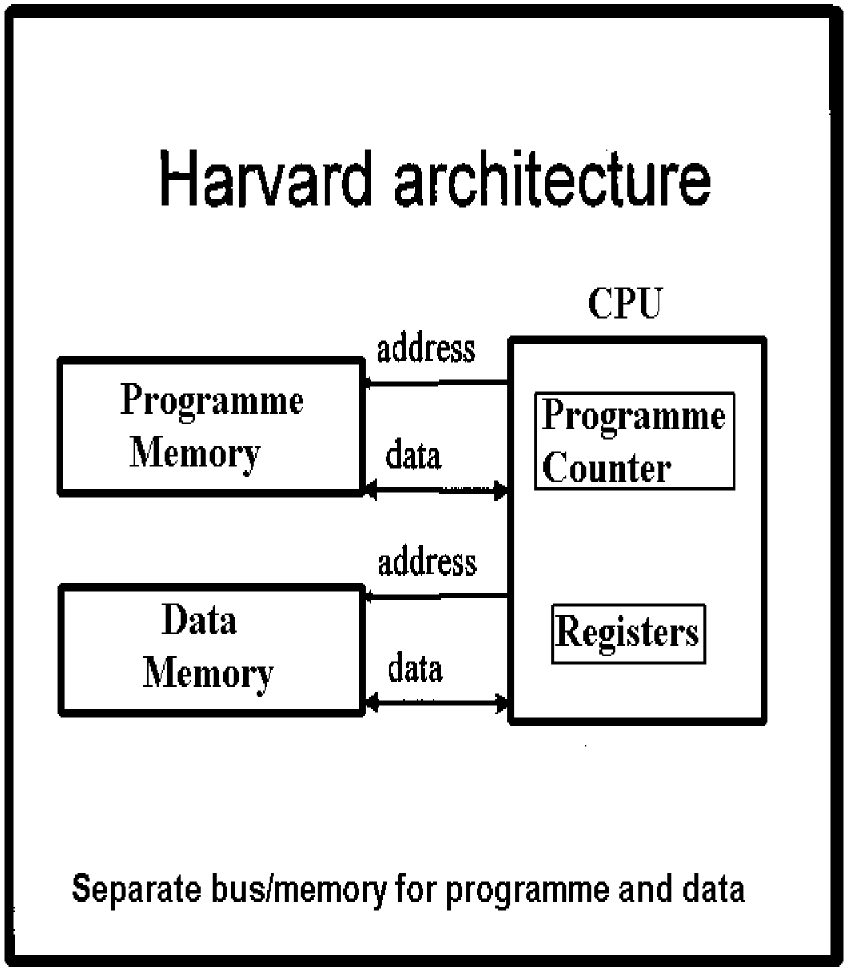
- Other aspects

Comparison of microcontroller with microprocessor



Computer Architectures:





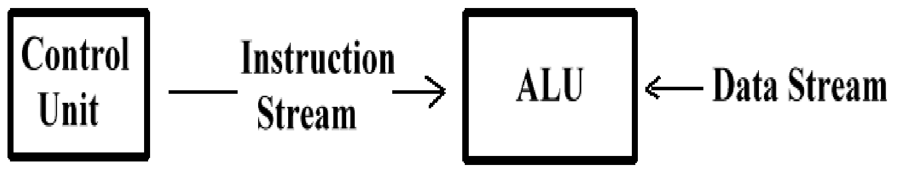
Most microprocessors use von Neumann architecture as Harvard would need many more pins to access two external buses.

However, more processing efficient Harvard Architecture with two buses easily implemented internally within a microcontroller.

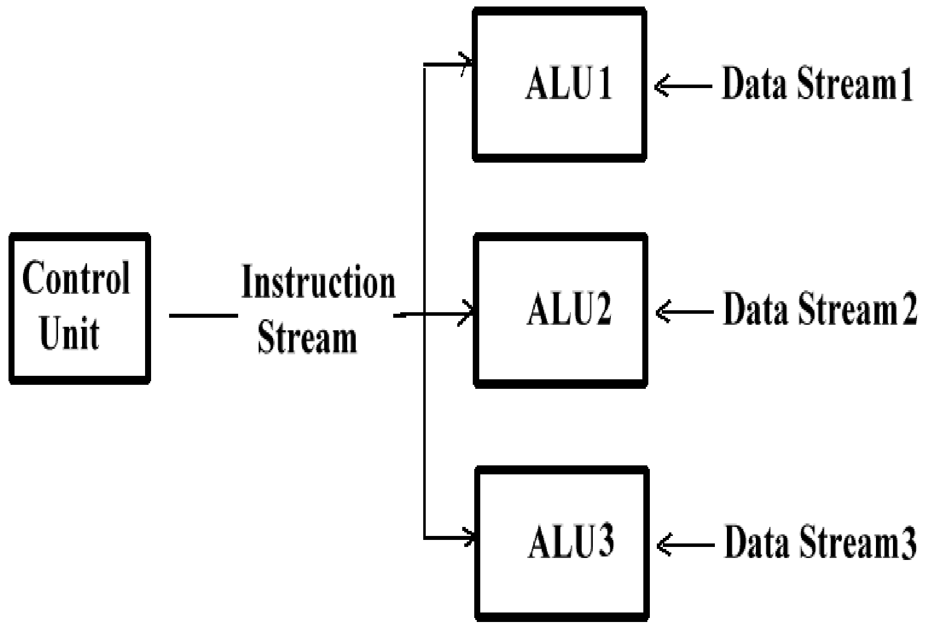
Multi-Processor / Parallel Processing

Flynn's Classification

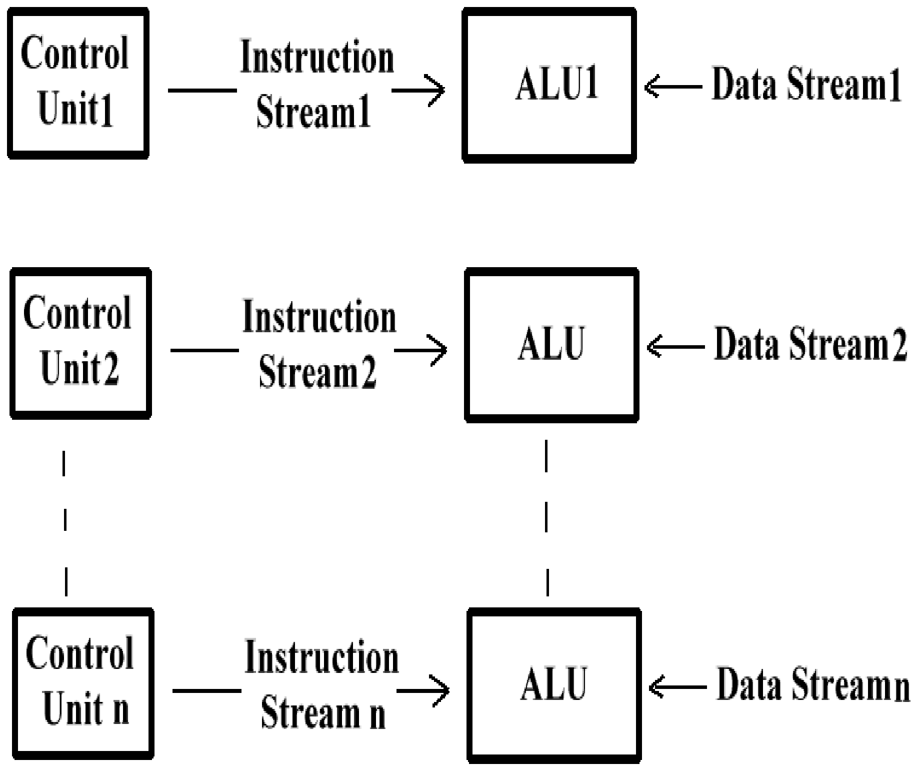
Single Instruction Stream, Single Data Stream(SISD)

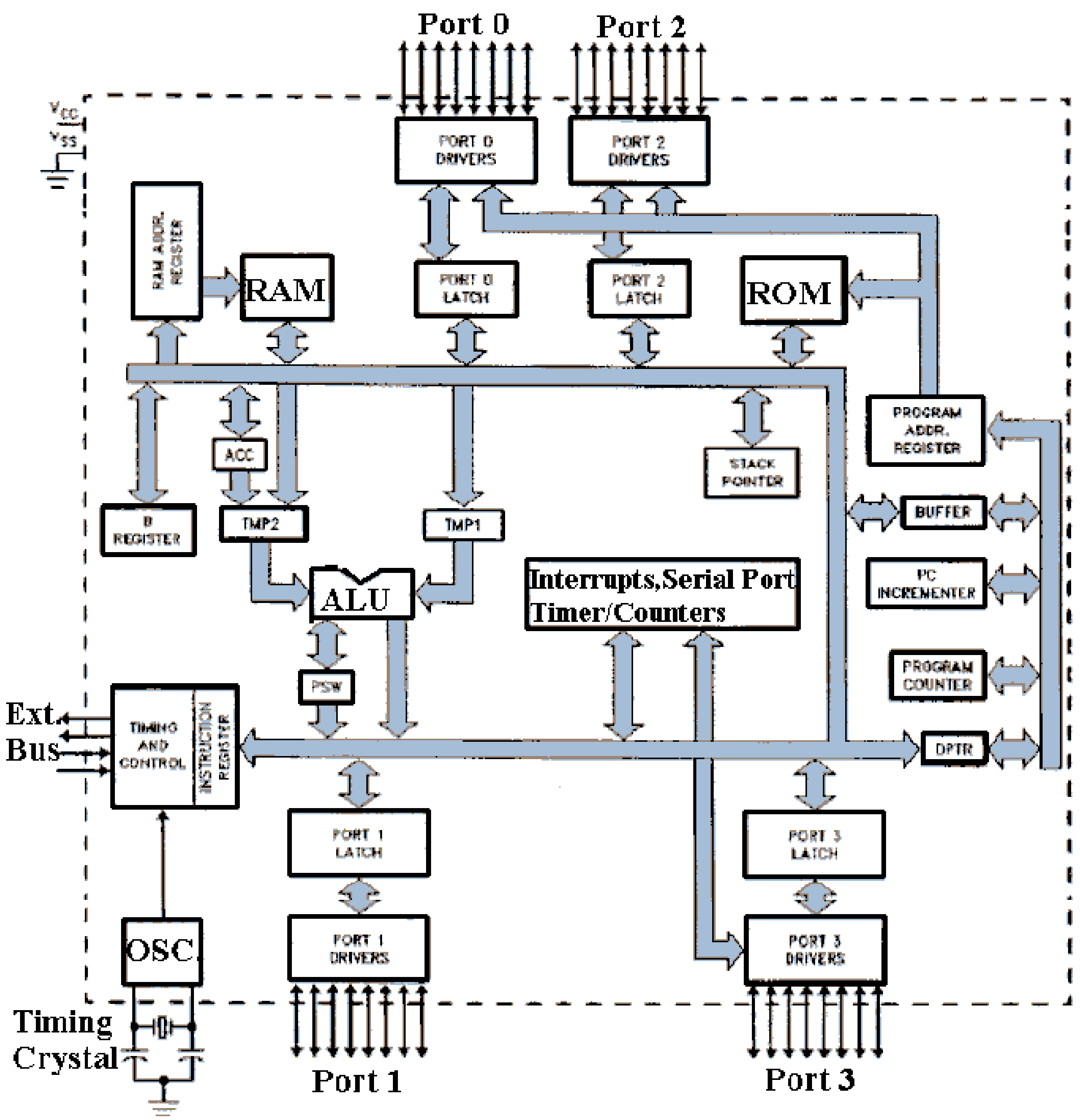


Single Instruction Stream, Multiple Data Stream(SIMD)



Multiple Instruction Stream, Multiple Data Stream(MIMD)





Microcontroller Example. An industry standard : 8051

Don't worry too much about complicated schematic on left. Mainly to show here the very many features included within one chip. Main ones are highlighted: Ports0-3, RAM, ROM, ALU, Oscillator, serial port, timer/counter, etc

8051 chip Includes:
- central processor
- ROM & RAM
- 3 counter/timers
- 4 parallel ports
- 1 Serial port

Requires only crystal for clock and Vcc.

Ports can be used to expand ROM & RAM bus externally

Some Microcontrollers

8bit microcontrollers:

Device	Pins(I/O)	Counters	RAM	ROM	Features
Intel 8051	40(32)	2	128	4k	Serial I/O Ext. memory
Microchip PIC	18(12)	1	32	512	RISC
Motorola 68HC11	52(40)	2	256	8k	A/D Watchdog
Texas TMS370	68(55)	2	256	4k	Ext memory A/D, Serial, WDT
Zilog eZ80190	100(46)	6	8k	0	Multiply, Serial Ext Memory16M

16bit microcontrollers:

Device	Pins(I/O)	Counters	RAM	ROM	Features
Renesas/ Hitachi H8/3032	80(63)	5	2k	64k	Ext Memory 1M Serial, 8x ADC PWM
Intel MCS-96 (family)	68(40)	2	232	8k	Ext Memory 64k Serial, ADC, WDT, PWM
Texas MSP430	64(55)	3	256-1k	4k-32k (OTP)	RISC-like Serial, ADC, LCD, low power

Very many types & manufacturers produce various versions with different facilities, e.g:

a) **Speed**: reduce
→1 clock / instruction

b) Memory
Data RAM upto 2kbyte
Programme ROM
upto128kbytes+
Types: EPROM, Flash, EEROM.

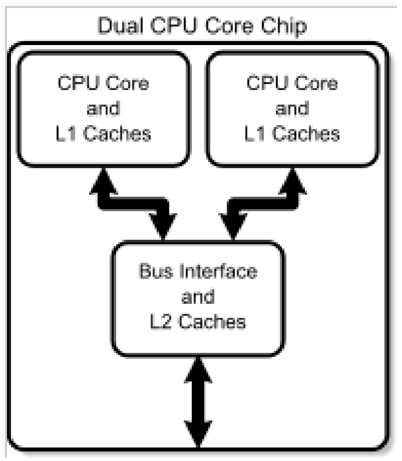
c) Communications
RS232
I^{2}C,
1-wire
CAN bus,
Ethernet, etc

d) Peripheral Drivers,
LCD, etc

Multi-Core and Parallel Processing

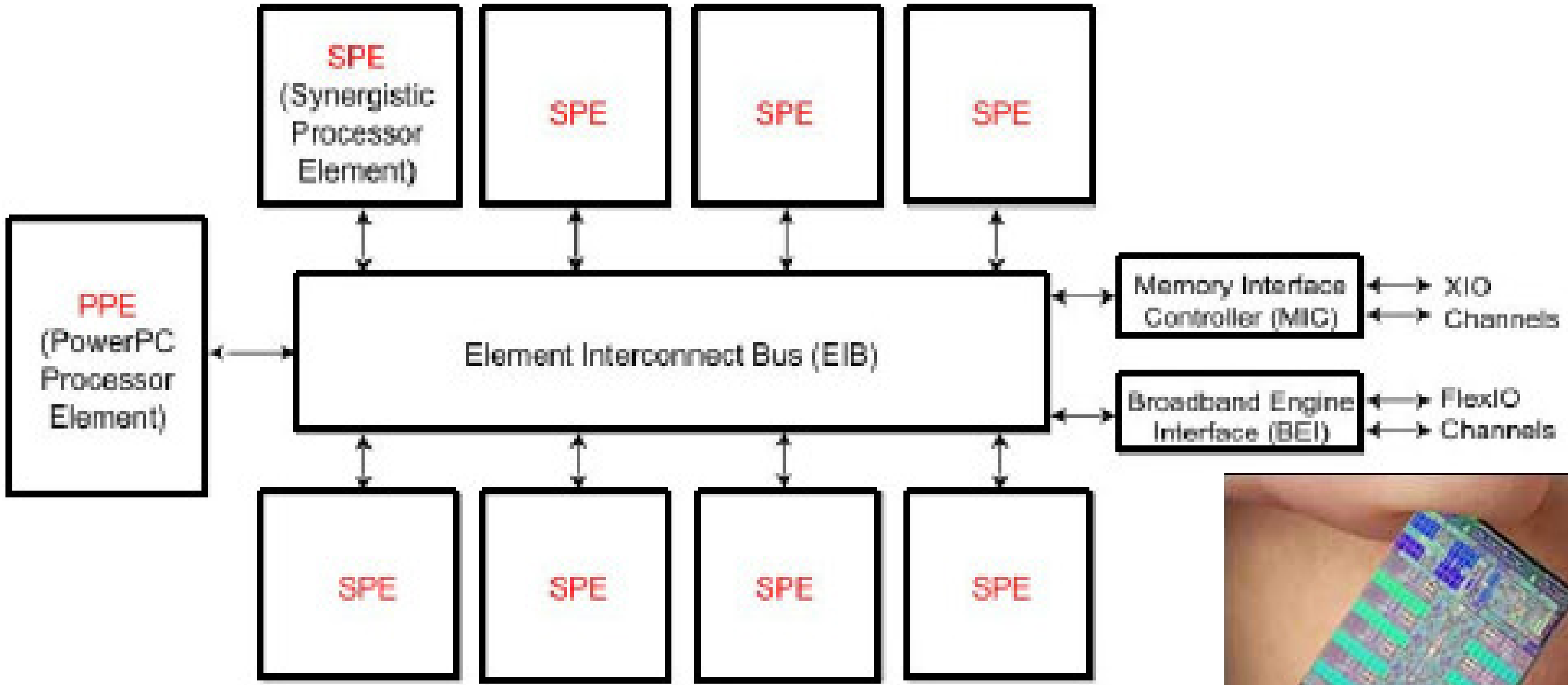
- Recent progress in processor speeds curbed by power dissipation problems. Every transistor switch action has I*V. Very many transistors at any one time often doing nothing but still dissipate heat!
- Over-clocking say from 3GHz to 3.6GHz possible – beyond manufacturers specifications → running hot reduces component lifetime→ plan redundancy use 10year guarantee computer for only 4year
- Supercomputers- similar problems. Germanium Arsenide logic + exotic components + liquid cooling → faster clocks.
- Supercomputer progress only through vastly parallel machines with many processors (groups of 1000's of PCs) - massively parallel hardware and applications
- Desktop PCs co-opted supercomputer model → 2 to 4 CPU cores on a single die. 'Dual core' , 'Quad core'

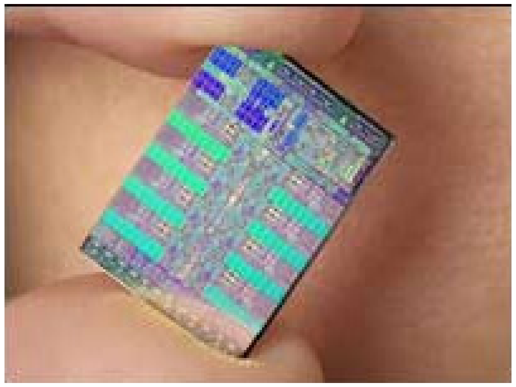
Intel Core 2 Duo   64bit dual core



- Typical 2 CPUs each with own L1 cache, share single L2 cache, that accesses single external bus to off-chip memory   (Cache memory- see later)
- But typically only ~1/3 of presently written PC programs can be parallelised
  → only 50% speedup as go from single → dual core
  (not the expected 100% - Amdahl's law)
- Single bus from L2 to off chip external RAM memory still a bus bottleneck
- Programs only fast as long as work from L1.
  But L1 size typically only 32-64kbytes! Small programs!, L2 typically 4Mbytes
- Multi-core (2 -4) use is presently optimised by operating system: multiple programs written for single core. Primarily for speeding up Multi-tasking, Multiple threads
- Moving towards Many-cores   >>4,
- Many-cores → really need to program specifically for parallel processors,
  need effective parallel languages, auto-parallelising compiler → Active Research Area

CELL Broadband Engine multi-processor for Playstation3


SPE
(Synergistic Processor Element)
SPE
SPE
SPE
PPE
(PowerPC Processor Element)
Element Interconnect Bus (EIB)
Memory Interface Controller (MIC)
XIO Channels
Broadband Engine Interface (BEI)
FlexIO Channels
SPE
SPE
SPE
SPE



Overview:

9 processor elements on a single chip:

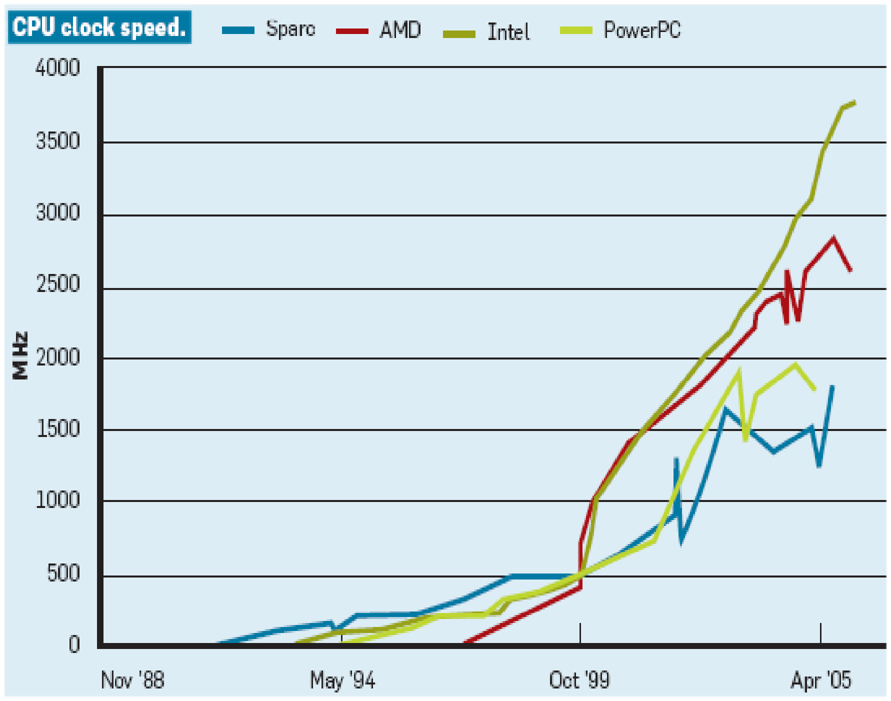
1 x 64 bit PowerPC processor element (PPE) optimised for operating system/control

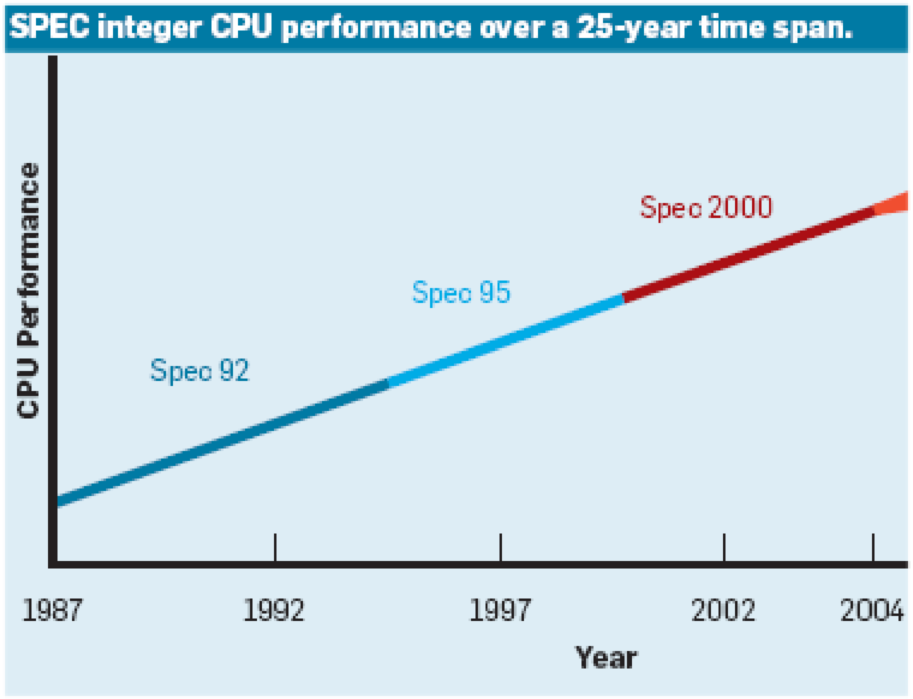
8 x Synergistic processor elements (SPE) optimised for compute intensive applications

PPE access main storage via load/store to private register file. Operating system neutral

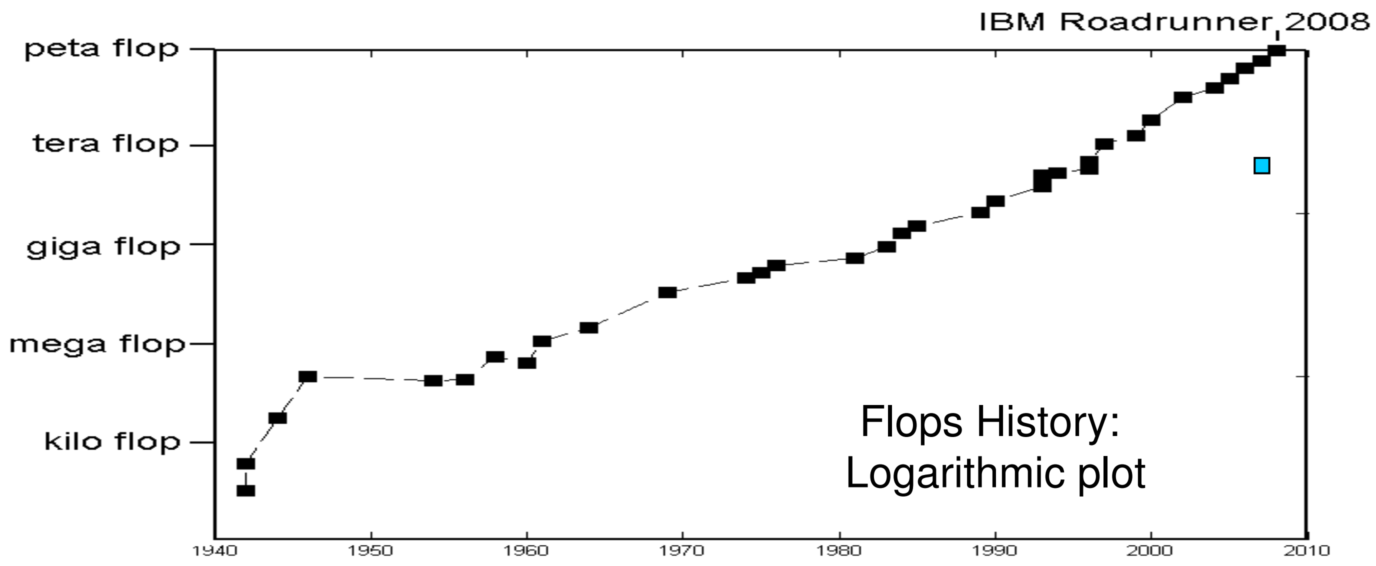
SPE access main storage via DMA to local memory for data & programme

See: http://www-01.ibm.com/chips/techlib/techlib.nsf/products/Cell_Broadband_Engine





Xera flop 10^{27}
Yotta flop 10^{24}
Zetta flop 10^{21}
Exa flop 10^{18}
Peta flop 10^{15}
Tera flop 10^{12}
Giga flop 10^{9}
Mega flop 10^{6}
Kilo flop 10^{3}



Today: 1) Top of the range PC Quad-core 30Gigaflops; 🟦

2) Roadrunner super computer (~ 130kcores = 13k Cell processors[9core]+ 7k AMD dual core) → 1.7 Petaflop

Future/Now: Processors as ‘CORES’ in FPGAs
Field Programmable Gate Arrays

Gate arrays- a sea of uncommitted logic cells can be configured as complex system that includes several microprocessors.

For example, a single Xilinx Virtex 4 family FPGA chip can include

- Two PowerPC 32bit RISC processors
- 192 DSP slices (multiply-accumulate units- for signal processing)
- 4 x 10/100/1000 Ethernet interfaces
- 142K Logic cells
- Block RAM

etc

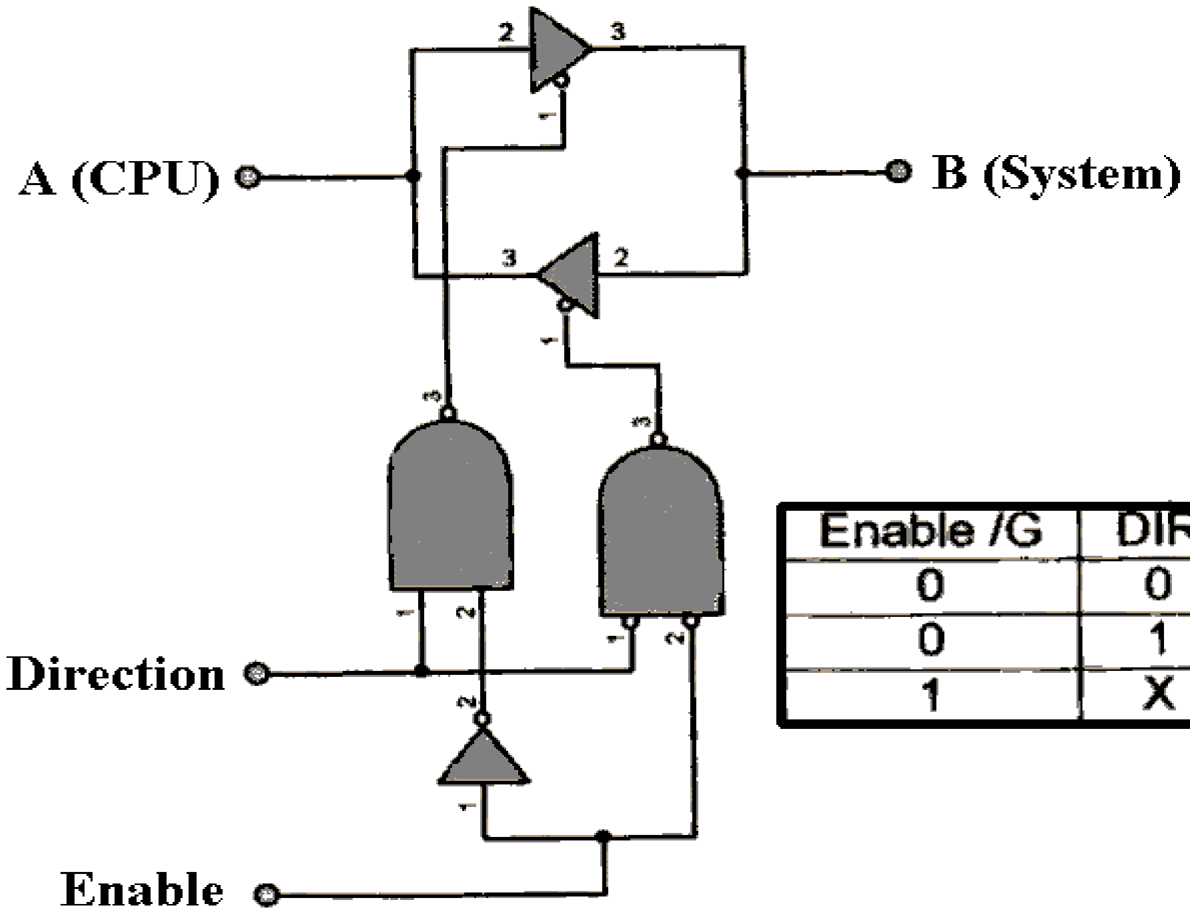
FPGAs can be re-configured in application to provide various functionalities. e.g. mobile phone, GPS receiver, etc.

FPGA cores often operate at lower voltage than data buses, often mixed voltage buses…. therefore there is a need to convert logic levels between buses…..

Multi-Voltage Level Systems

Systems often utilise more than one voltage level logic to optimise the system by making use of the most appropriate chips. Various logic level standards: 5V, 3.3V, 2.5V, 1.8V, 1.5V, etc.
Lower voltage → faster + lower power dissipation
Need bidirectional bus transceivers with voltage level shifting between different voltage buses.

Example below where, say, bus A is 5V logic and bus B is 3.3V logic.
Chip with 8 or 16 data lines, each connected as shown here.
Buses can be isolated or joined by ~*Enable* line, while data left-right direction is set by *Direction* line.



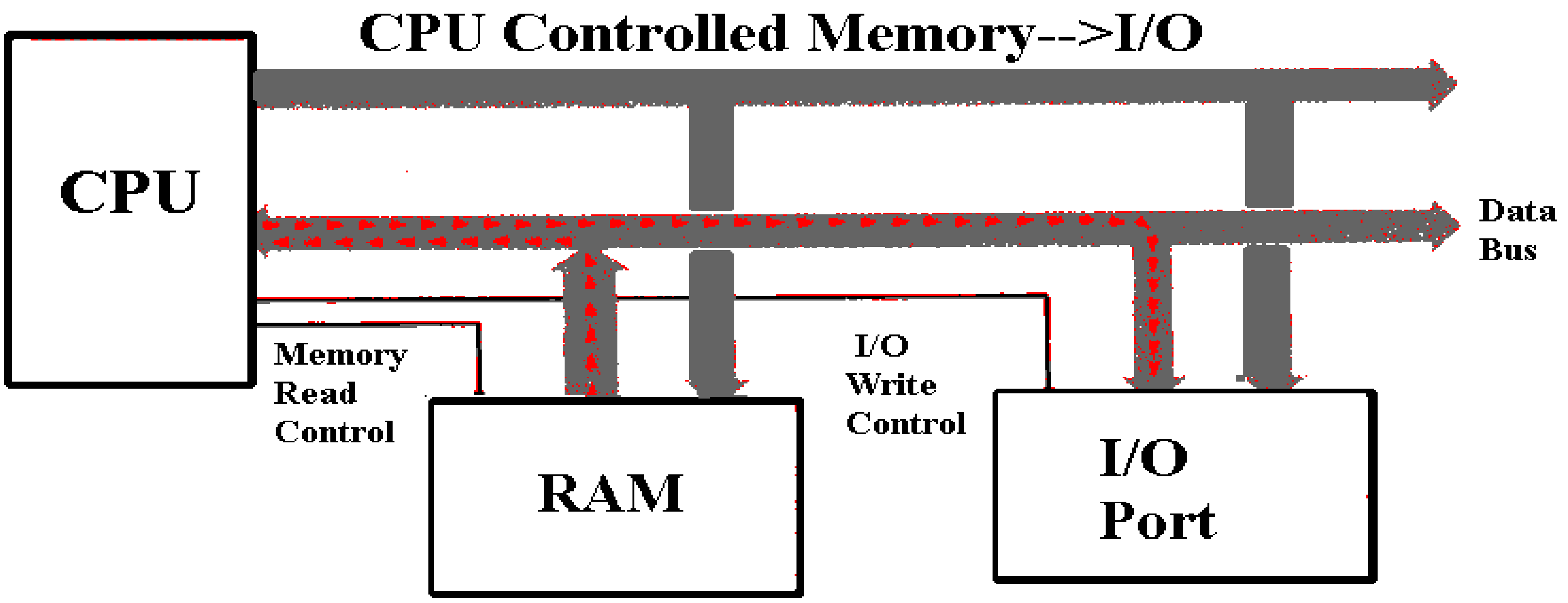
Enable /G	DIR	Operation
0	0	B>>A
0	1	A>>B
1	X	Hi Z

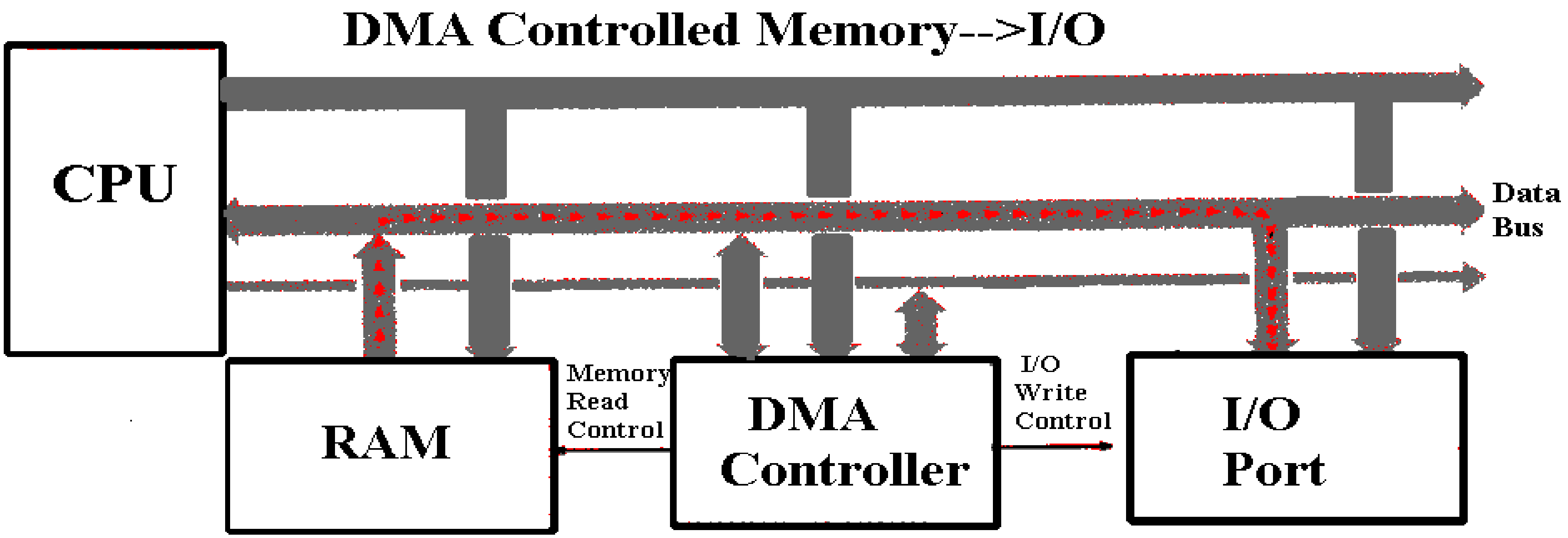
Part 8.

Other System Aspects:

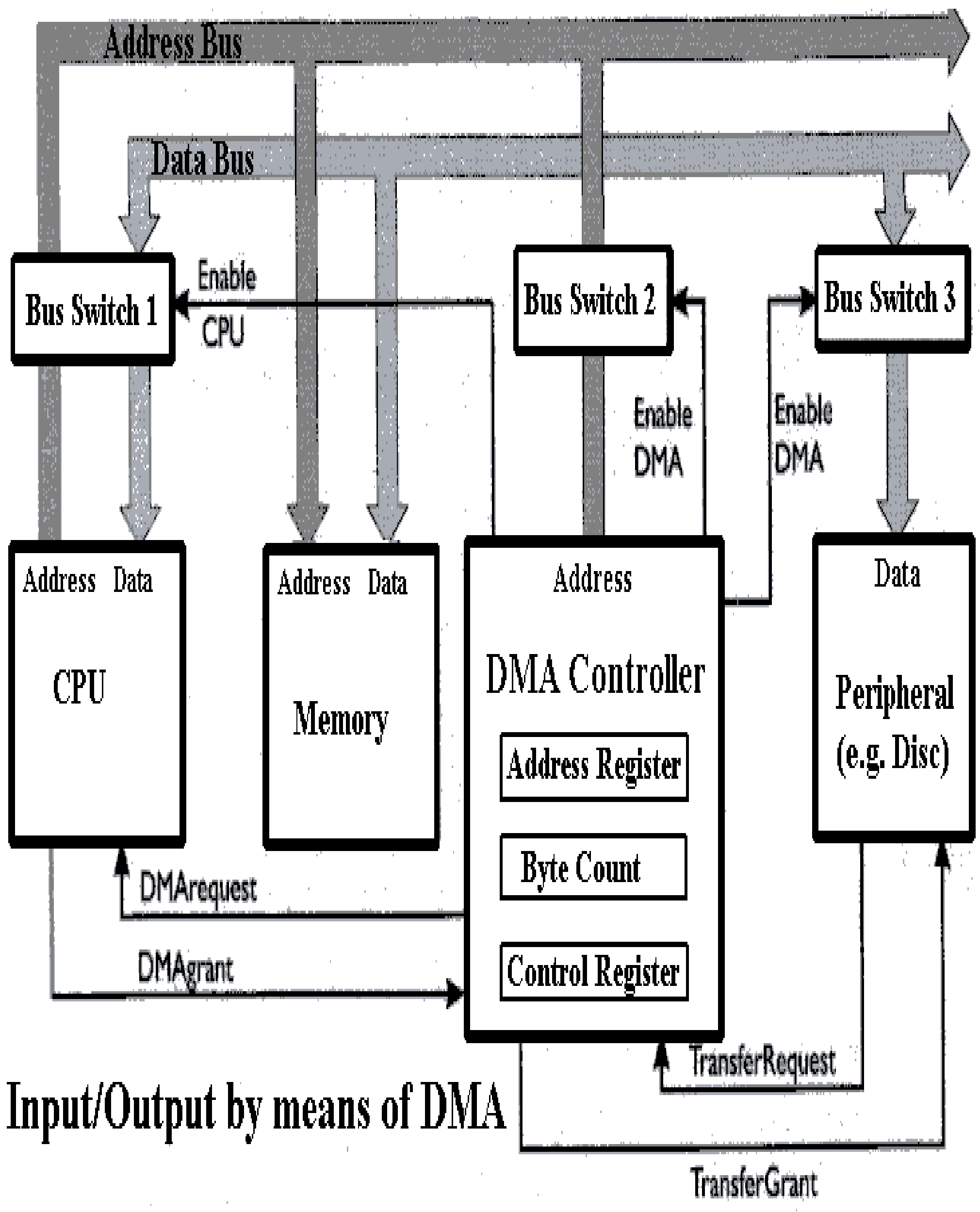
- Direct Memory Access
- Cache memory
- Operating systems & Multi-tasking
- Connecting to sensors & actuators
  I^2C, 1-wire, CAN
- Programming, cross compiling,
  system debug

Direct Memory Access, DMA: for Fast I/O ←→ Memory

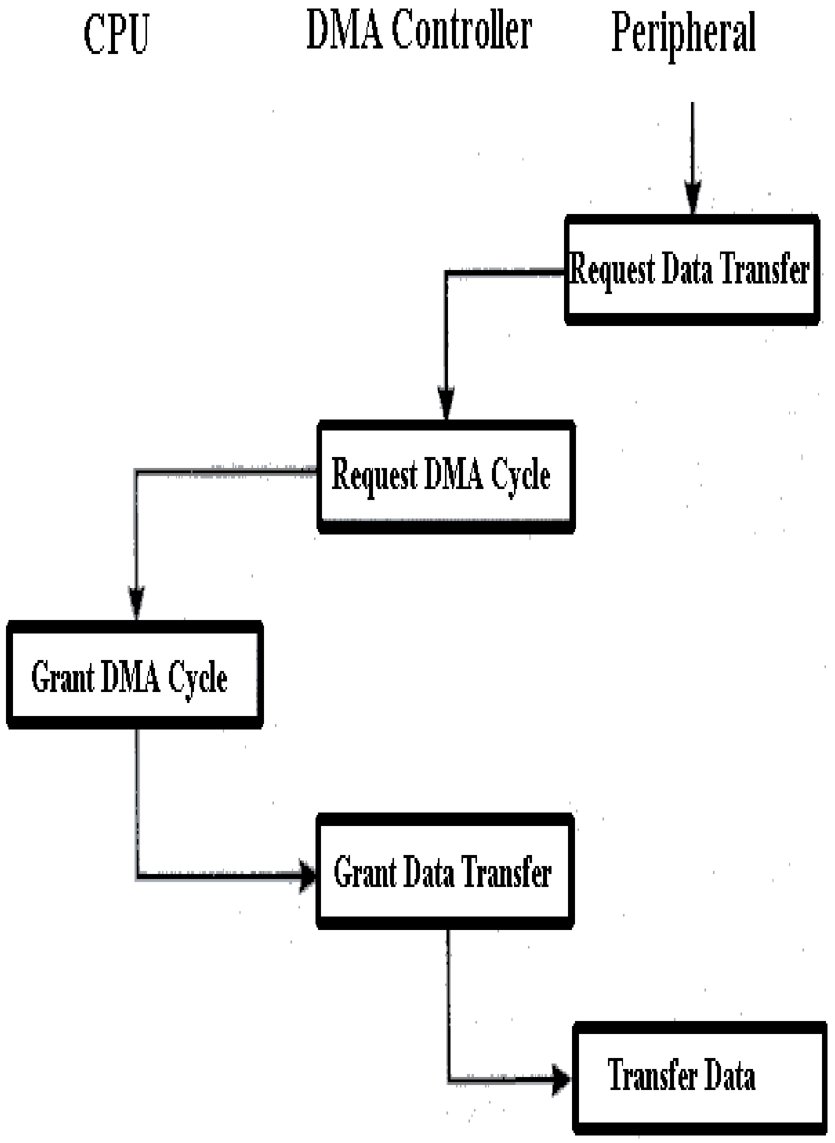




Direct Memory Access, DMA, cont'd

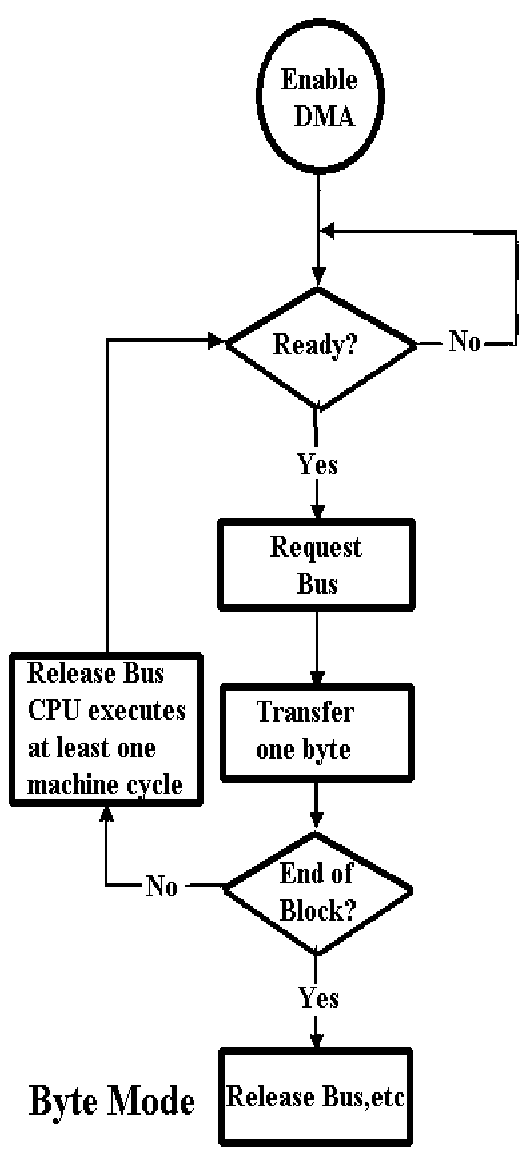


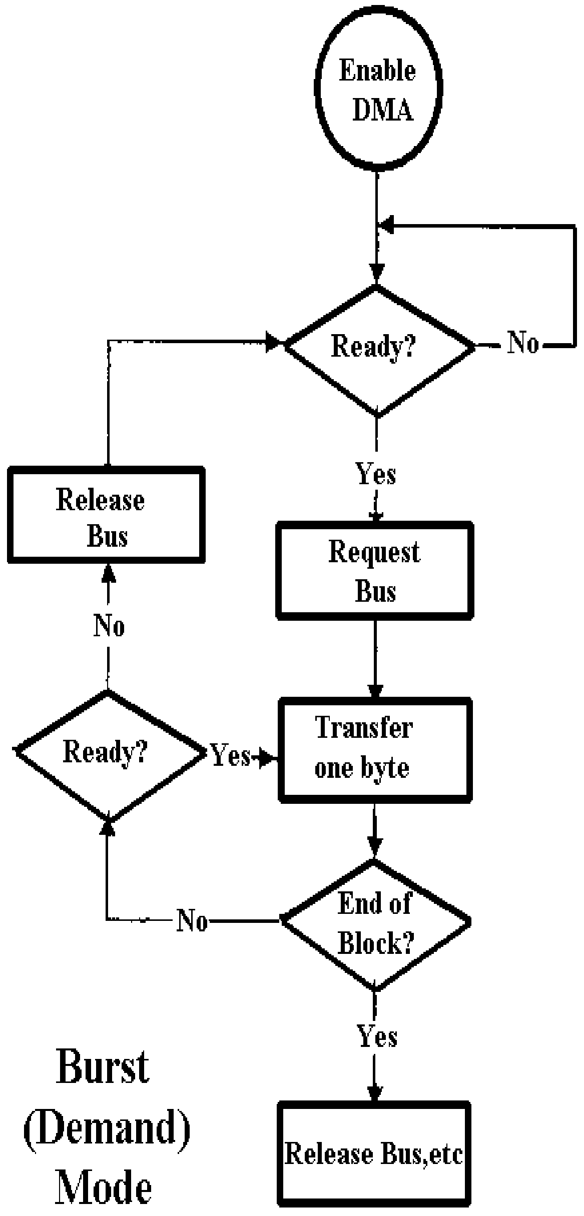
Input/Output by means of DMA

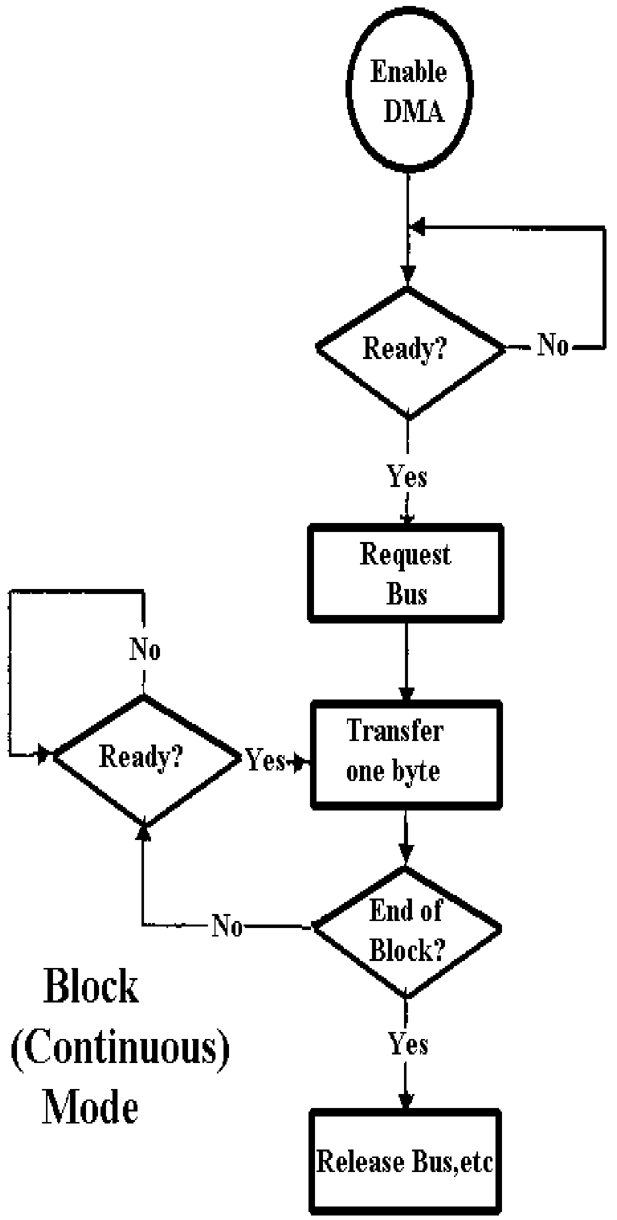


Protocol Flowchart for DMA Operation

DMA modes:





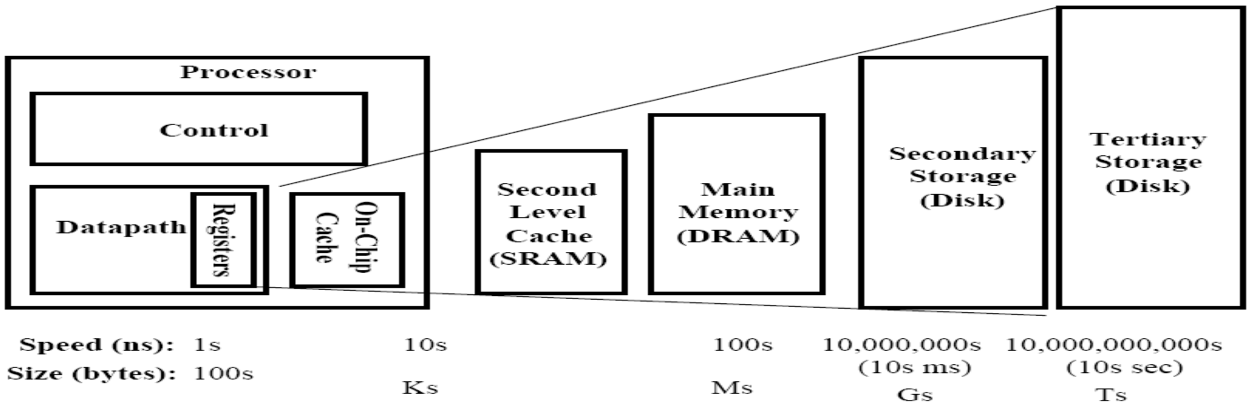


'Generous!' 'Reasonable' 'Greedy!'

CACHE MEMORY For Faster Programs

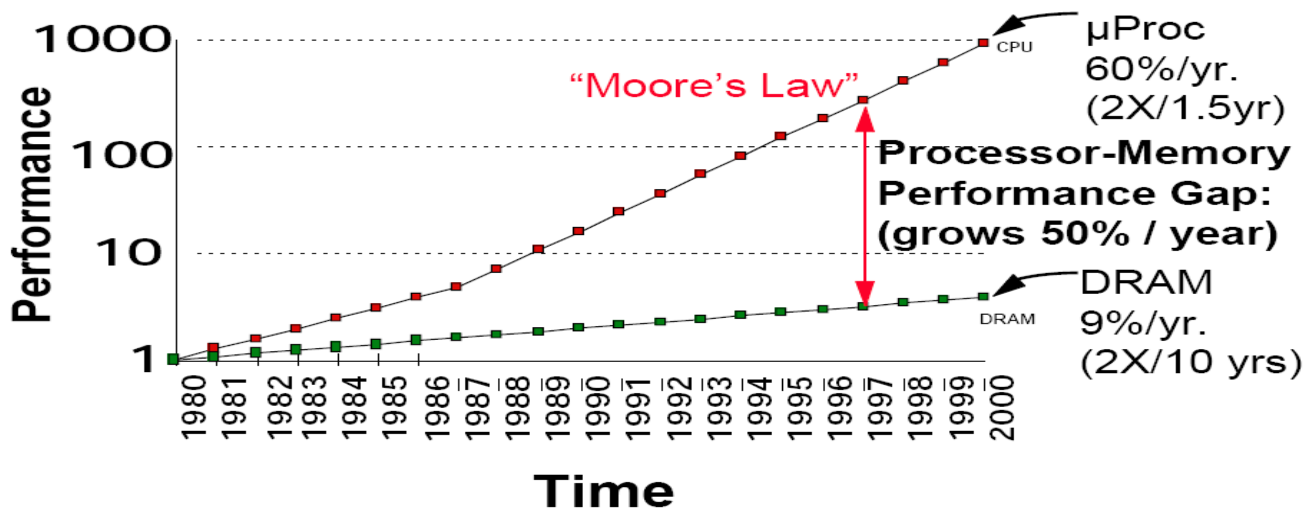
Memory Hierarchy of a Modern Computer System

- Present the user with as much memory as is available in the cheapest technology.
- Provide access at the speed offered by the fastest technology.

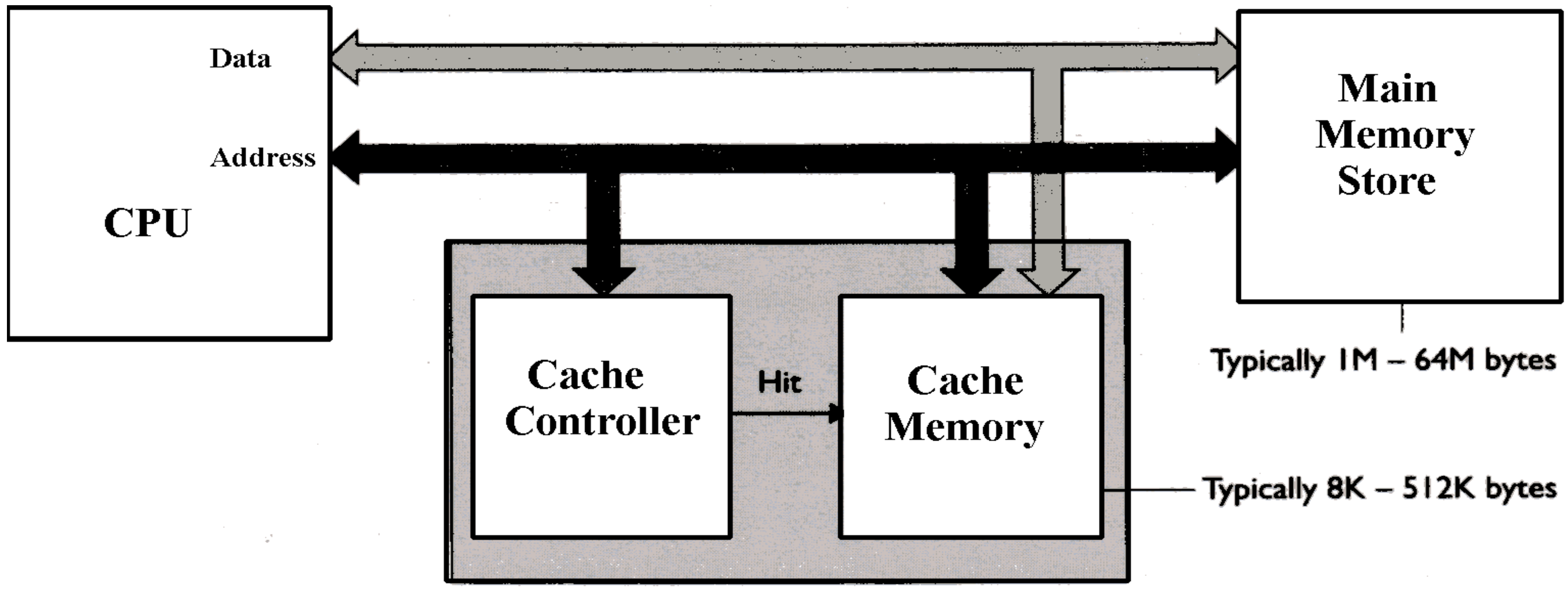


Who Cares About the Memory Hierarchy?

Processor-DRAM Memory Gap (latency)



Memory Cache for faster programmes



Cache memory – local fast memory used to hold pre-fetched operation codes

Speed-up depends on (i) ratio of cache memory speed to main memory speed,
(ii) how often op-code is already in cache (a hit),
(iii) average number of machine/clock cycles per instruction

Cache controller needs to (a) ‘look ahead’ in programme to fetch instructions.
(b) keep address tags of instructions to identify ‘hits’

Note that the 68000 uses a standard simple 2-word pre-fetch, absorbing some program op-code fetch cycles within execution cycles as many clock cyles/instruction.

The principle of locality

Temporal locality means it's likely to be referenced again soon

Spatial locality means nearby items are likely to be needed soon

(Analogy: books on a library desk)

- Locality is clear for **instruction** reads:

Temporal: Programs have lots of loops.

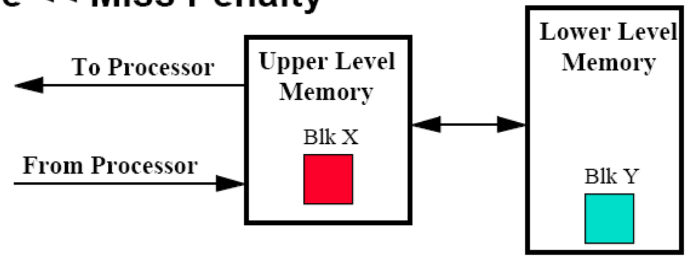
Spatial: Most of the time instrs accessed sequentially.

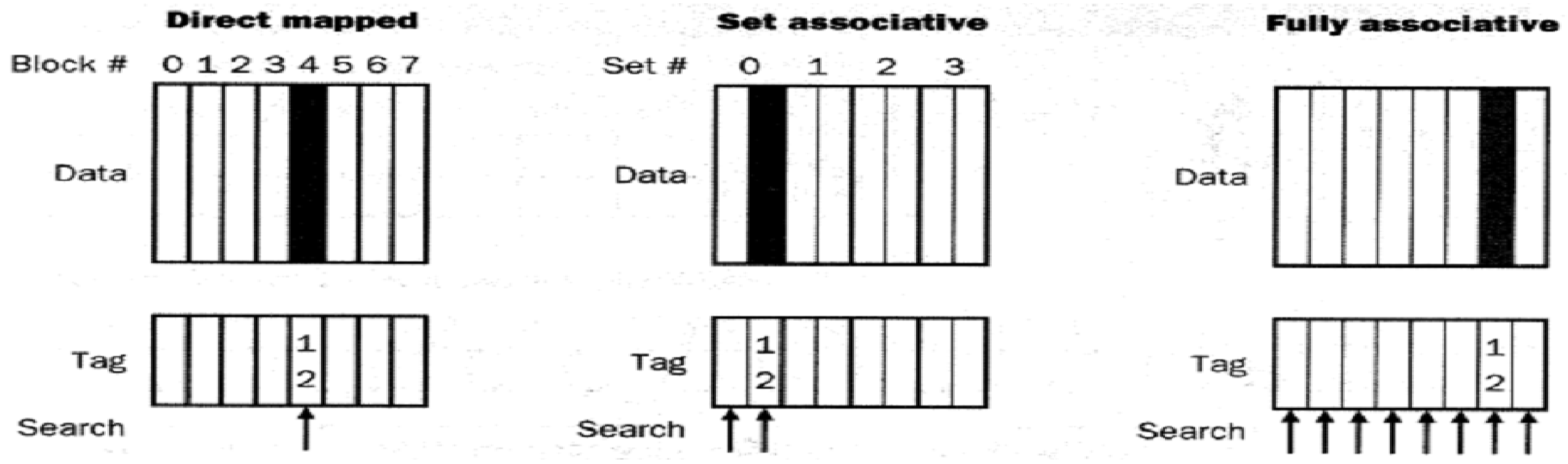
and it happens only a bit less for **data** read/write:

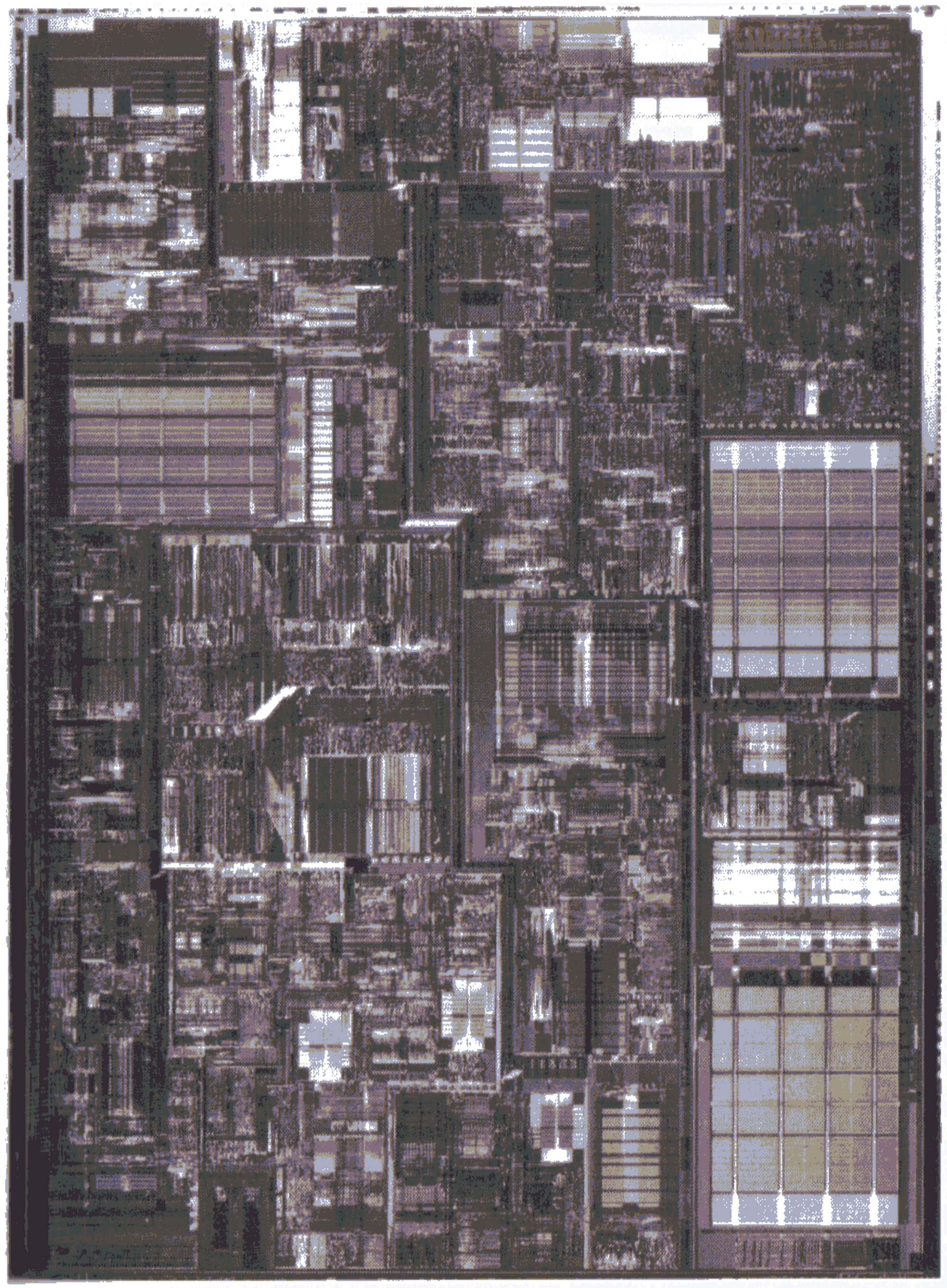
- Parts of a data structure mostly takes up contiguous chunks of memory.
- A program tends to make several accesses to a particular data structure in quick succession.

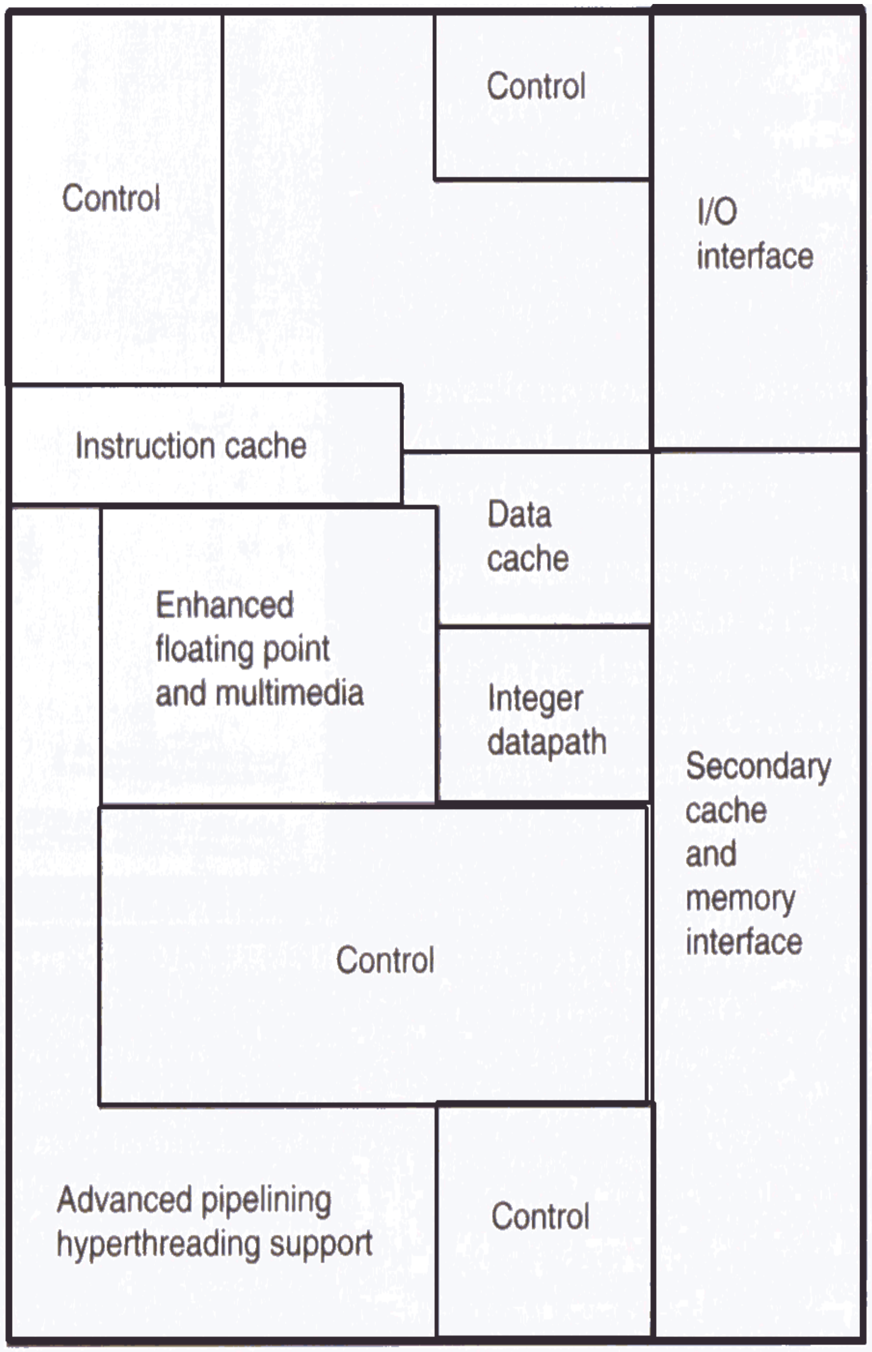
Memory Hierarchy: Terminology

- **Hit**: data appears in some block in the upper level (example: Block X)
  - **Hit Rate**: the fraction of memory access found in the upper level
  - **Hit Time**: Time to access the upper level which consists of RAM access time + Time to determine hit/miss
- **Miss**: data needs to be retrieve from a block in the lower level (Block Y)
  - **Miss Rate** = 1 - (Hit Rate)
  - **Miss Penalty**: Time to replace a block in the upper level + Time to deliver the block the processor
- Hit Time << Miss Penalty

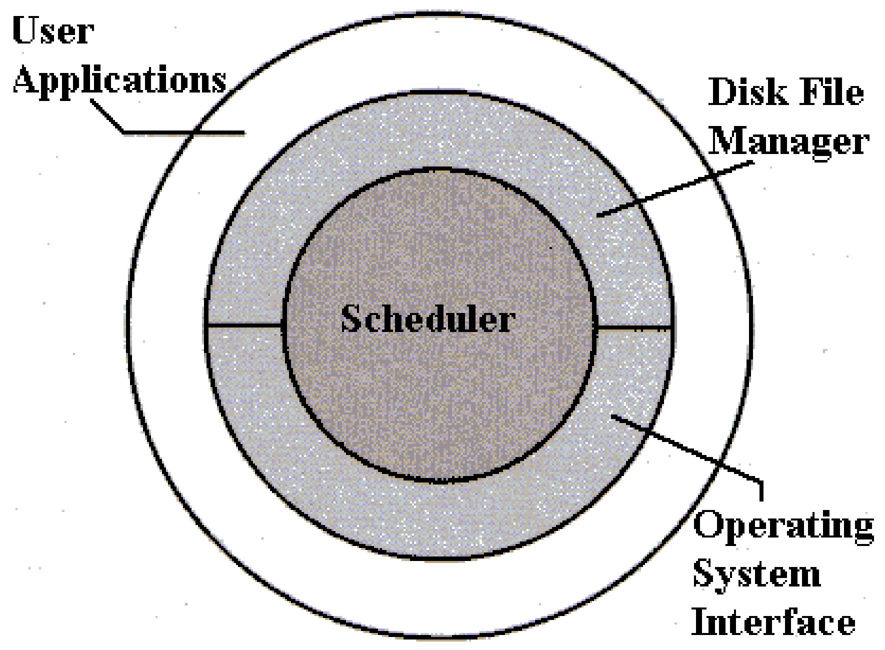






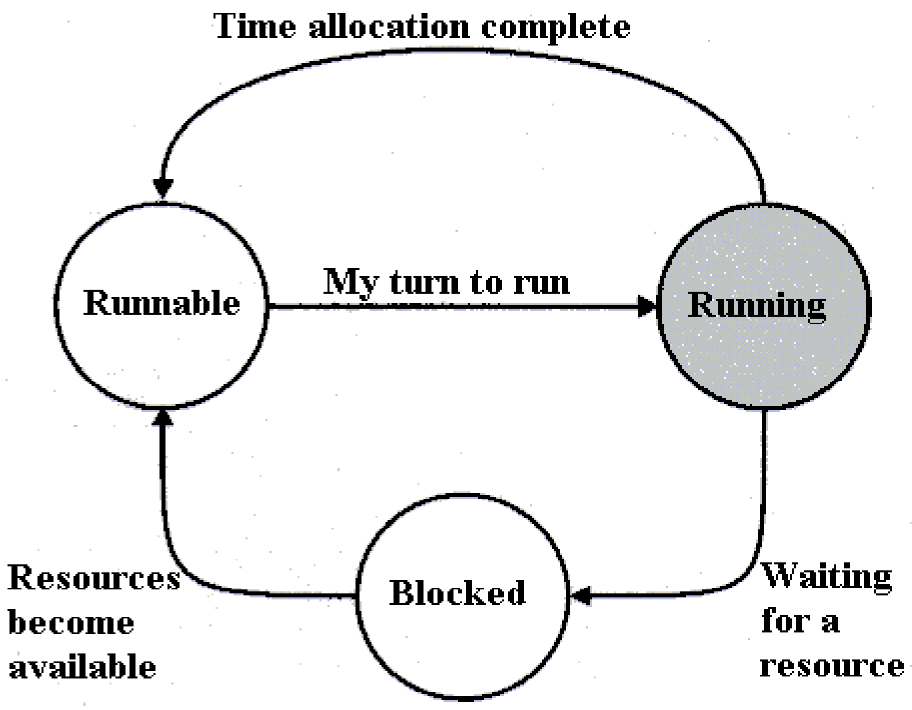

Control
Control
I/O interface
Instruction cache
Data cache
Enhanced floating point and multimedia
Integer datapath
Secondary cache and memory interface
Control
Advanced pipelining hyperthreading support
Control

The left-hand side is a microphotograph of the Pentium 4 processor chip, and the right-hand side shows the major blocks in the processor.



Operating System

- Overall OS: Co-ordinates, optimises efficiency, schedules tasks (processes).
- Applications use resources provided by OS
- OS hides details of the hardware.

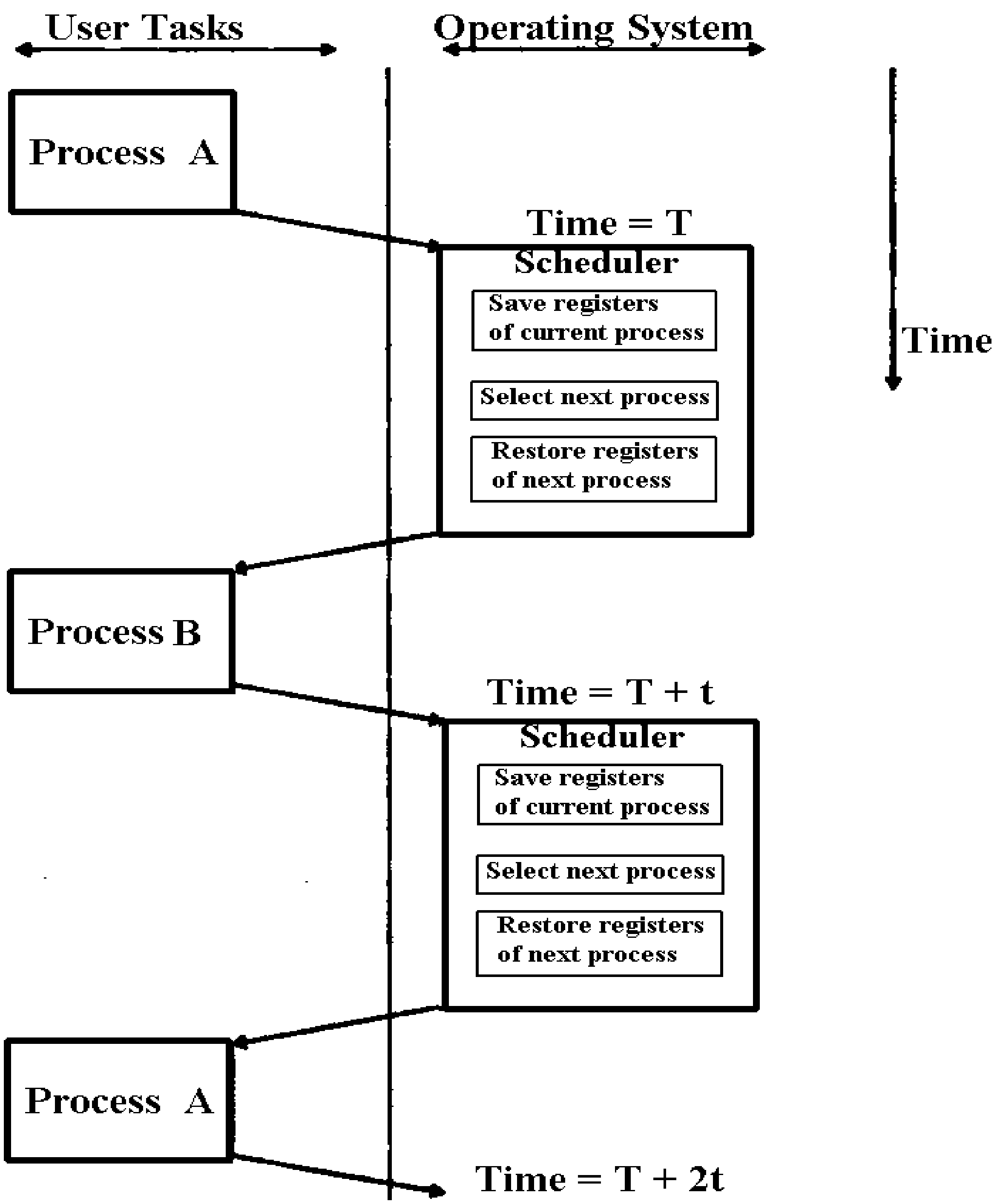


Task Scheduling:

Each process is in one of three states:

- Runnable: available & waiting
- Running: running now
- Blocked: waiting for an essential resource to become available.

Multi-Tasking



OS safely switches contexts between processes.

Scheduler saves current process's context (volatile portion) and invokes a new Process.

Present state of each process must be saved at end of running it.

Previous state of each process must be restored at the start of running it again.

Use separate stack areas for each process to save register status.

Connecting to other systems

Previously we mentioned RS232, USB, Firewire, SCSI, etc as main standards for microprocessor / computer connection to peripherals.

Main standards for microprocessor/controllers networking to sensors/actuators:

- **CAN, Controlled Area Network**

  Balanced 2-wire interface with differential line drivers / receivers (like RS485)
  Used in Automobile, Transport & Industry for up to 100m communications.
  e.g. Automotive Bus, Industrial Field Bus

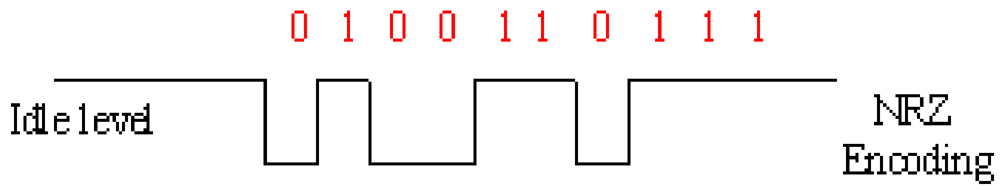
- **I^{2}C,**

  Fast 2-wire bus, up to 400kbits/s

- **1-wire,**

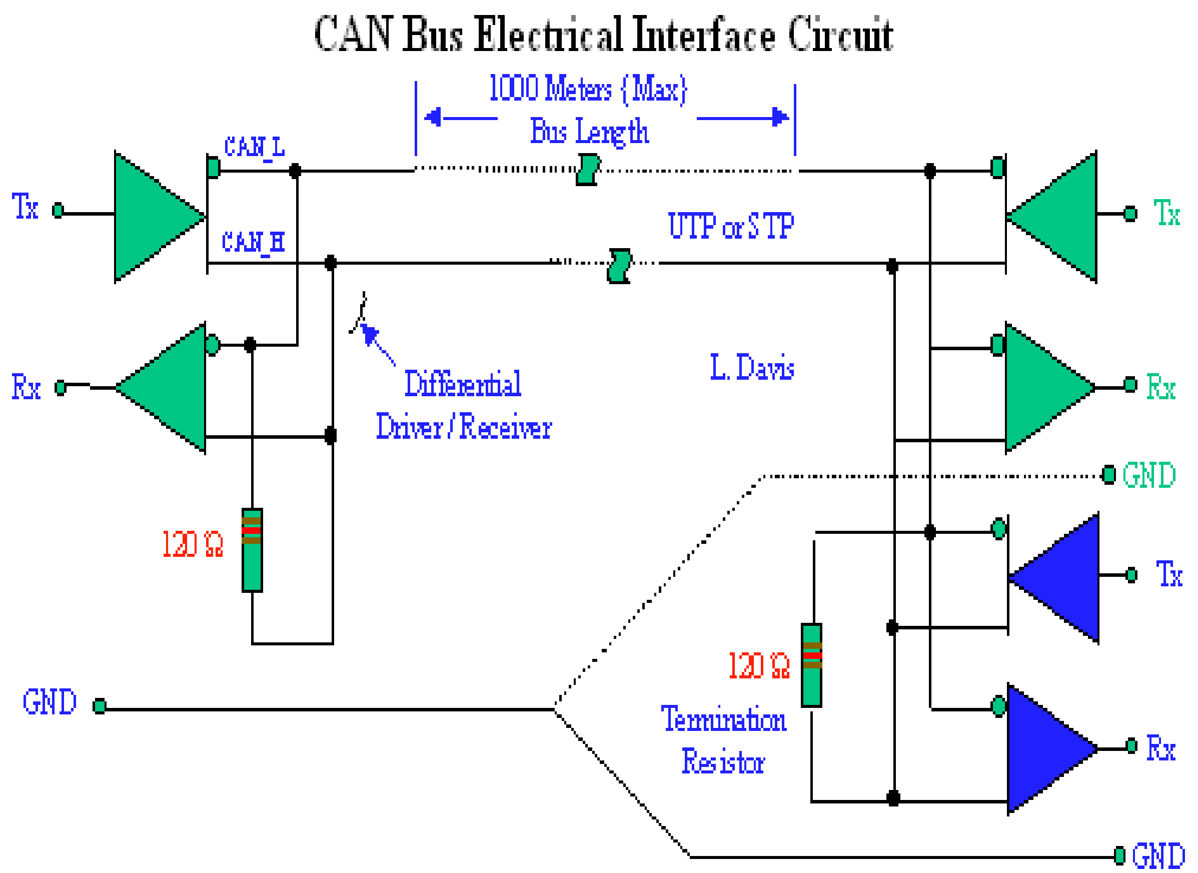
  Single wire used by master to communicate with slaves, also used to power slave devices. Economic in hardware resources. Ideal for short distances.

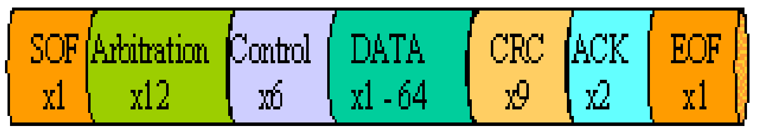
CANbus



Each byte transmitted as Non-Return to Zero, NRZ, asynchronous, with start & stop bits (like RS-232.).

Balanced 2-wire interface with differential line drivers & receivers in parallel (like RS422/RS485).





CAN bus data frame, bytes

Data sent in frame:

Start of Frame

Arbitration Control: 11-29 bits determine priority of message, arbitrates between devices.

Data: 0-8 bytes of data

Cyclic Redundancy Check: 15 bit checksum

Acknowledge: any CANbus device receiving acknowledges TX retransmits if none.

End of Frame

I^2C Bus

Each device on bus has unique address. Multi-master- more than 1 device can control bus. Arbitration between contending devices.

Serial 8bit data. Two wire bus shared by all devices: SDA Serial Data line;SCL Serial Clock Line

Example:

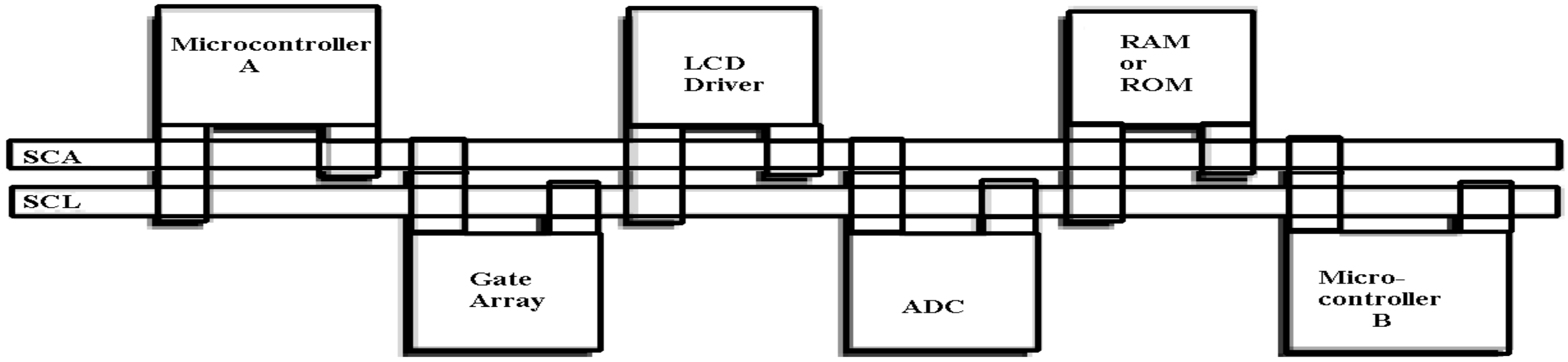
1) Suppose microcontroller A wants to send information to microcontroller B:

- microcontroller A (master), addresses microcontroller B (slave)
- microcontroller A (master-transmitter), sends data to microcontroller B (slave- receiver)
- microcontroller A terminates the transfer

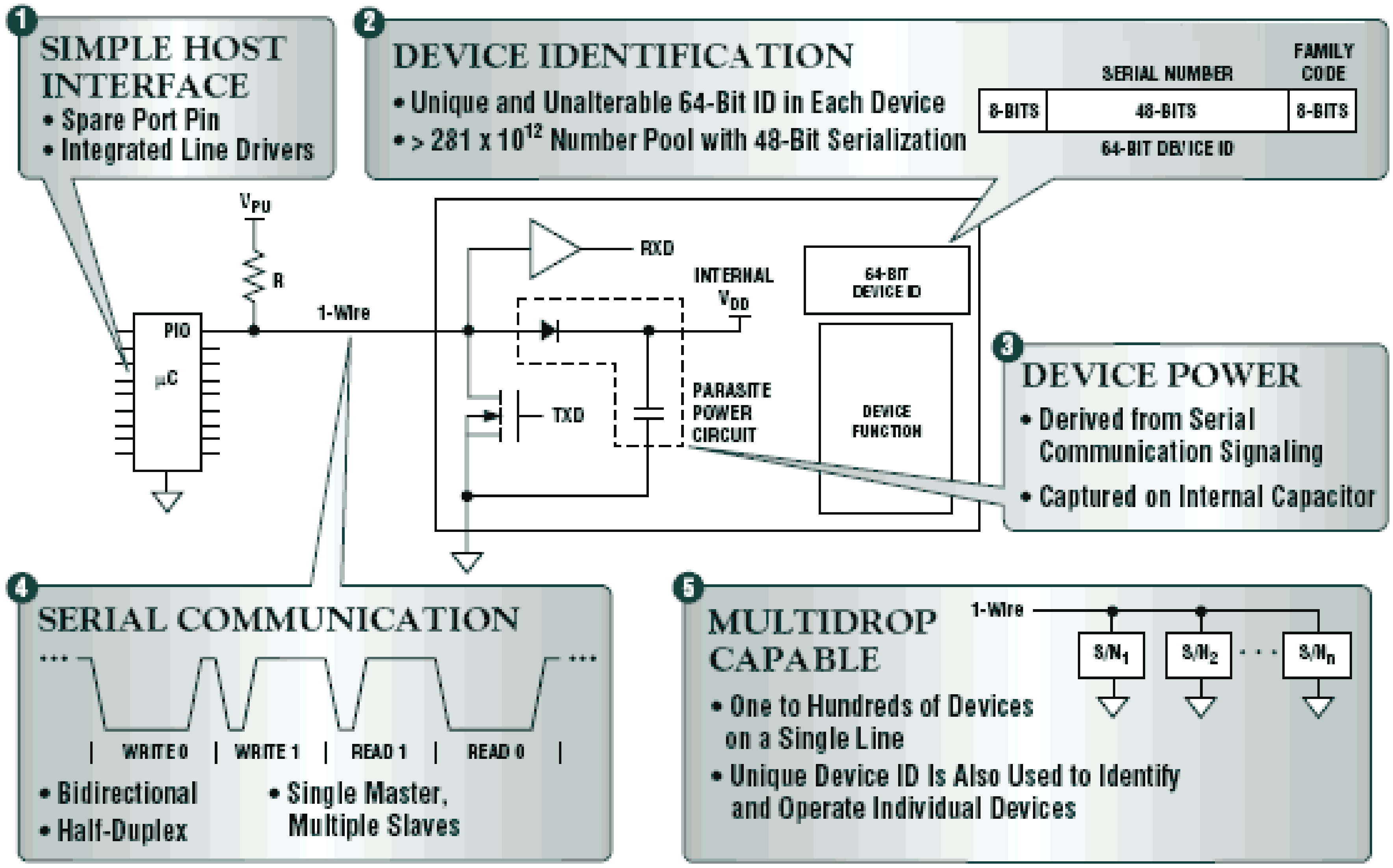
2) If microcontroller A wants to receive information from microcontroller B:

- microcontroller A (master) addresses microcontroller B (slave)
- microcontroller A (master- receiver) receives data from microcontroller B (slave- transmitter)
- microcontroller A terminates the transfer.

the master (microcontroller A) generates the timing and terminates the transfer.



1-wire   (Maxim-Dallas)



1 SIMPLE HOST INTERFACE

- Spare Port Pin
- Integrated Line Drivers

2 DEVICE IDENTIFICATION

- Unique and Unalterable 64-Bit ID in Each Device
- $> 281 \times 10^{12}$ Number Pool with 48-Bit Serialization

3 DEVICE POWER

- Derived from Serial Communication Signaling
- Captured on Internal Capacitor

4 SERIAL COMMUNICATION

- Bidirectional
- Half-Duplex
- Single Master, Multiple Slaves

5 MULTIDROP CAPABLE

- One to Hundreds of Devices on a Single Line
- Unique Device ID Is Also Used to Identify and Operate Individual Devices

Device families include: ADC, DAC, Analogue Switches, Memory, Temperature Sensors, etc.

Programming Microprocessors/Microcontrollers

Directly in Low Level Assembler Language.

- Slow, tedious, unforgiving, only practical for small systems
- Timing for critical programme loops and for interfaces can be set precisely from number of clocks/intruction.
- Memory use/allocation can be easily organised/kept within bounds.
- Better to understand what is actually happening in fine detail.
- Difficult to appreciate the whole design.

Indirect via Cross Compiling from Higher Level Language, e.g. C

- Relatively quick, easy to implement in C, often necessary for large systems
- Difficult to ensure the precise timing of critical parts.
- Care must be taken in memory use and data variable type assignment.
- Easy to appreciate the whole and verify overall design functionality.

In practice overall system often written in a higher language with some time critical sub-systems written directly in the relevant assembler language.

System Debug Tools

- **ROM Emulator.(software development in real system)**

  Replace programme ROM in its socket by lead to ‘PC’ which provides code CPU and remainder of target embedded system as is. Allows code to be run from any given address up to user selectable breakpoints.

- **In-Circuit Emulator, ICE (real in-system verification)**

  Extract CPU from socket (or attach adaptor & tristate CPU) and replace by ICE system. ICE provides as for ROM emulator + more control: permits full view of system, signals and bus devices with full trace facilities.

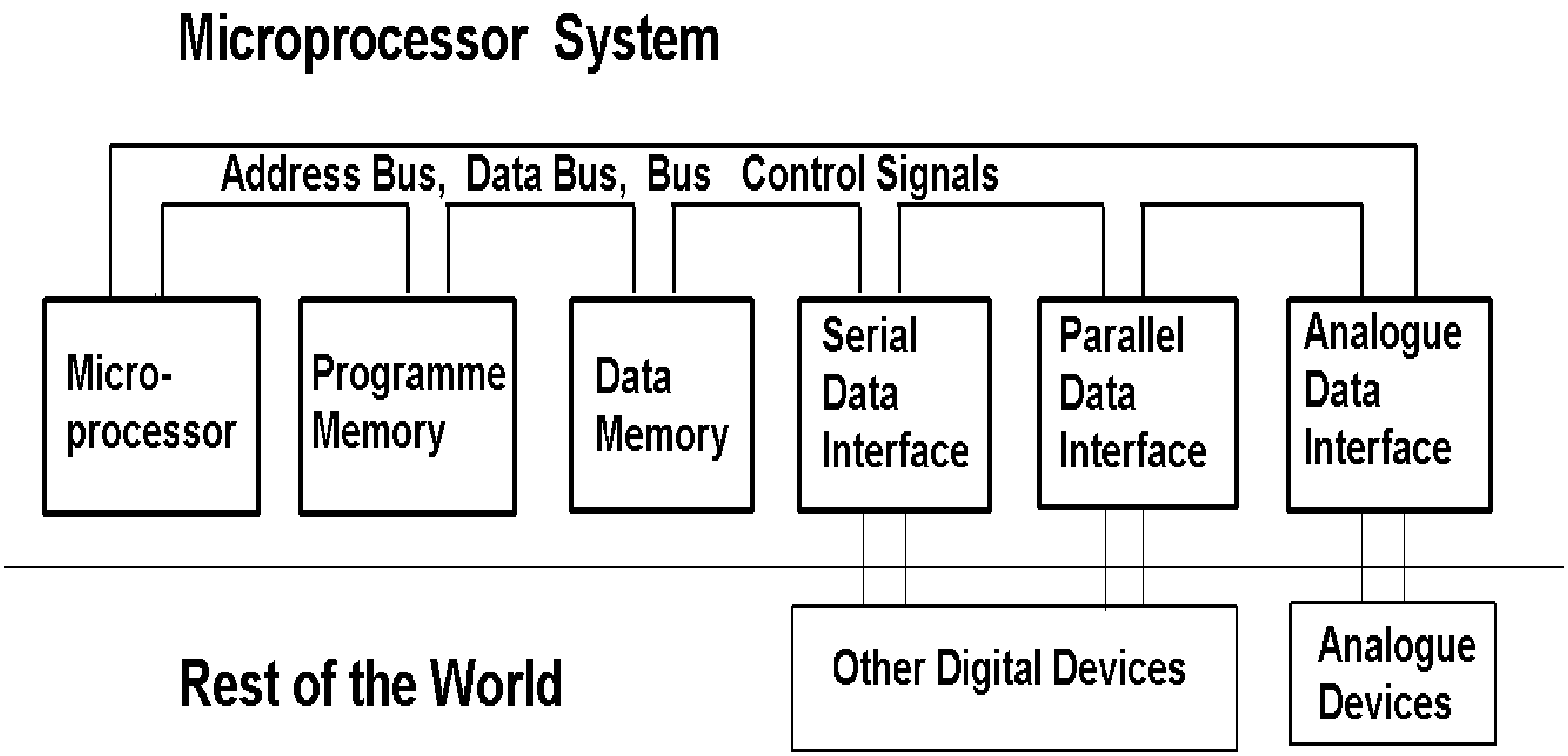
- **Logic Analyser (real system hardware debug)**

  Multi-channel, multi-signal digital oscilloscope / monitor, specially for detailed analysis of system bus. Most useful for debugging specific problem events, e.g. system timing of communications between devices.

- **Simulator (virtual system)**

  Software simulation – no use of actual hardware.

Course Summary



Problem Sheet 1: Address Decoding

A 68000 microprocessor with address lines A_1-A_{23} is to be connected to memory chips:

a) Two ROM memory chips (each 2k x 16)
11 address lines A_0-$A_{10,}$ /ROM (chip select)

a) Two RAM memory chips (each 4k x 16)
12 address lines A_0-A_{11}, /RAM (chip select), /WRITE

The required address space is:

ROM	**0000H-0FFFH**
RAM	**1000H-2FFFH**

Outline the connections for :

a) Simplest linear address decoding, assuming no other devices are on the bus

b) Full address decoding, allowing for system expansion

c) Full address decoding as in (b) but when the above chips are not available and the only chips available are 2k x 8 ROMs and 4k x 8 RAMs

Work out for next week- we'll go over possible solutions in lecture

Problem Sheet 2 :   General Review

1) Draw and label the block diagram of a small microprocessor system that might be used in a dedicated application. What is the function of each section.

2) Outline the sequence of operations involved in the execution of a typical instruction by a microprocessor such as a 68000. Provide a labelled timing diagram.

3) Explain the differences between (a) operation-code (programme word) fetch cycle, (b) data memory read cycle, and (c) data memory write cycle.

4) What is meant by a ‘wait state’ and how does its use affect the microprocessor bus control signals and memory cycle times. What devices initiate wait states?

5) Discuss the operation and use of the programme counter and stack pointer and show how they control the sequence of programme execution.

6) Explain the difference between static and dynamic RAM. What are the different types of non-volatile memory?

7) A particular peripheral chip contains four internal address locations which may be written to or read from. Draw a diagram of the bus connections necessary to connect two such chips to a microprocessor using (a) linear addressing (b) fully decoded addressing.

8) Explain the operation of data latches and buffers. How are these used for microprocessor input and output ports?

9) What functions can be performed by counter-timer chips and how might these be used?

Problem Sheet 3 : General Review

1) Explain the time sequence of actions that occur when a subroutine is called from the main programme with particular reference to role of the stack pointer.

2) How does an external device interrupt the microprocessor programme? What is meant by interrupt priority and how does the microprocessor control which devices can cause interrupts?

3) Explain the difference between synchronous and asynchronous serial data communication and sketch a typical asynchronous data character. How would you obtain this signal from an 8bit parallel data byte?

4) How does a Universal Asynchronous Receiver/Transmitter interface with (a) external systems, and (b) with its host microprocessor?

5) Explain the operation of three types of digital to analogue convertors(DAC):
(a) Potential divider network DAC, (b)Binary weighted input DAC, and (c) R-2R ladder DAC.

6) With the aid of pseudo-code or flow charts explain microprocessor programme sequences that use an 8-bit DAC to generate the following analogue signal patterns:
(a) square wave whose amplitude is software controlled, (b) a square wave whose period is software controlled (c) a full amplitude sawtooth wave, and (d) a full amplitude triangular wave.

7) How might a microprocessor utilise a timer/counter chip, an input port, and a DAC to provide an analogue sawtooth wave whose period is controlled by an external digital input.

Problem Sheet 4 : General Review

1) Explain the following types of Analogue to Digital Convertors (ADC) operate:
(a) Successive approximation, (b) Flash ADC, and (c) Dual slope ADC.

2) What is meant by aliasing error? What does the Nyquist frequency signify?
What is the minimum sampling rate required for signals which contain frequency components up to 20kHz?

3) A microprocessor system has two ADC inputs, one DAC output, and an alarm bell operated by a single output port bit. The two ADC inputs monitor the voltage across an electric motor and the current taken by it.
Provide the pseudo-code, or flow chart, for a microprocessor programme that outputs an analogue value proportional to the electric power taken by the motor and also rings an alarm whenever the power taken exceeds a preset maximum value. (*hint: power = voltage x current*)

4) The motor of question (4) produces many short-term current spikes that cause false alarms.
Modify the pseudo- programme/ flow chart to reduce the effects of spikes by averaging over 8 successive current samples.

5) Describe a typical microcontroller and highlight its essential features, identifying their uses.

6) Choose a typical microcontroller application and show how the microcontroller is used.

7) What is Direct Memory Access (DMA) and why is it used?

8) What is Cache memory and what is it's purpose?

9) Provide a brief description of different types of microcontroller network: I^2C; CANbus; and 1-wire.

10) Outline the essential features of an operating system. What is meant by multi-tasking?