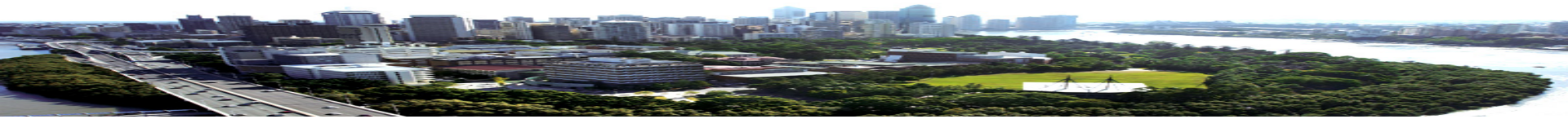
ENB 350 Real-time Computer based systems

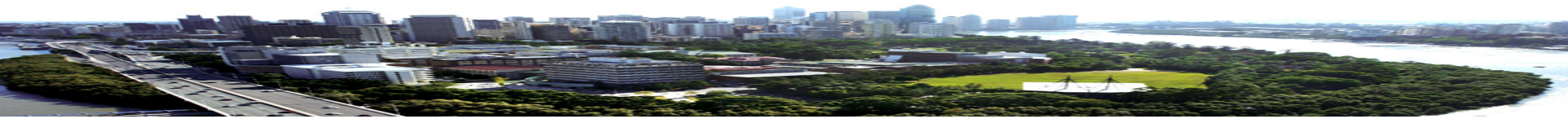
Lecture 2 – MICROCONTROLLER ARCHITECTURE

V. Chandran



Contents

- State chart representations
- CPU, Memory and I/O
- Registers
- Architecture - peripherals
- Assembly Instructions
- Debugging and examining values



Prerequisite Test Results

Max score : 98%

Min score : 18%

Total students : 46

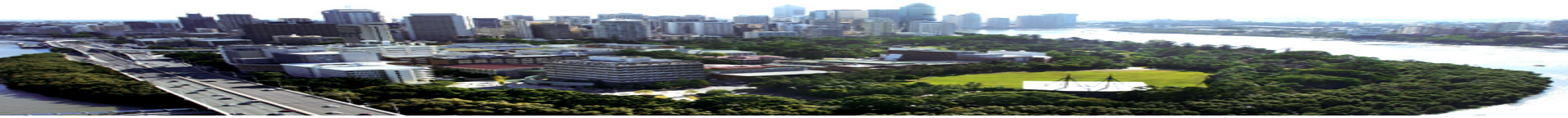
You know what you have missed and can revise appropriately.

PARTICULARLY:

Number representation and arithmetic

C features and usage

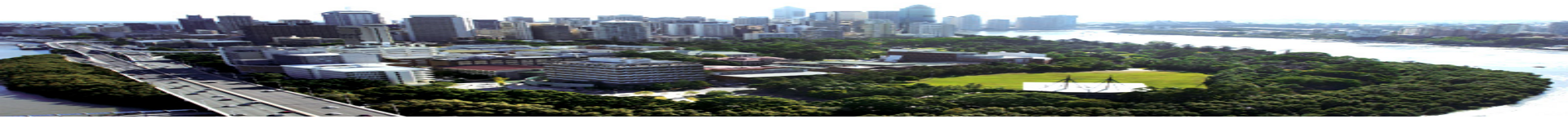
Question	%
Gates	70
CMOS output	78
Full adder	60
CPU components	64
Flip flops	52
Program Counter	54
2s complement	9
C operators	50
C keywords	67
C function	35



Questions

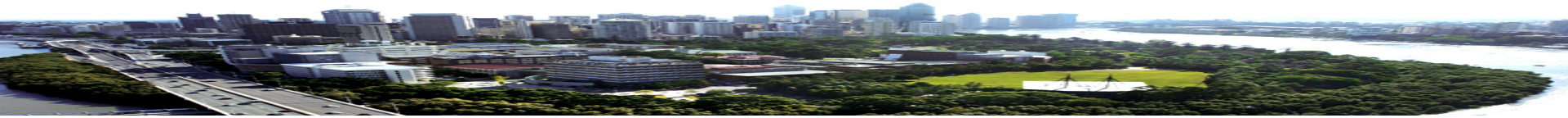
How do you interface a switch or digital device with a microcontroller?

How do you interface an analog device?



Requirements specification

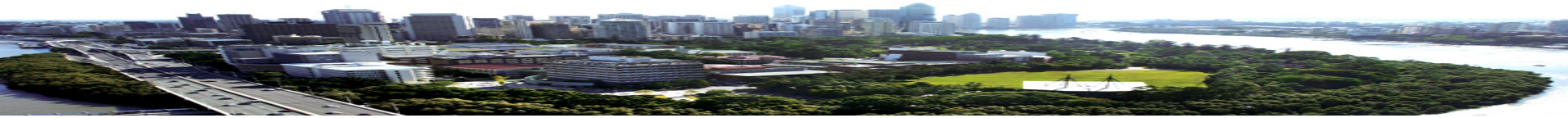
- Logical descriptions and simplification
  - Natural language and mathematical descriptions
  - Top down structure analysis, flow charts, state diagrams
  - Object oriented approaches
- Formal Verification
  - Predicate calculus
- Temporal requirements
  - Natural language and mathematical descriptions
  - Concurrency with state charts, Petri nets
  - The ‘Z’ language (1982) has been extended with RTIL (real time interval logic) to provide such features



Questions

What is a finite state machine? A state diagram? State charts?

(Reference book: Real-time systems design and analysis by Philip A. Laplante sections 4.4.3 and 4.4.4)



Finite State Machines (Moore)

$M = \{S, i, T, \sum, \delta\}$

S = set of states

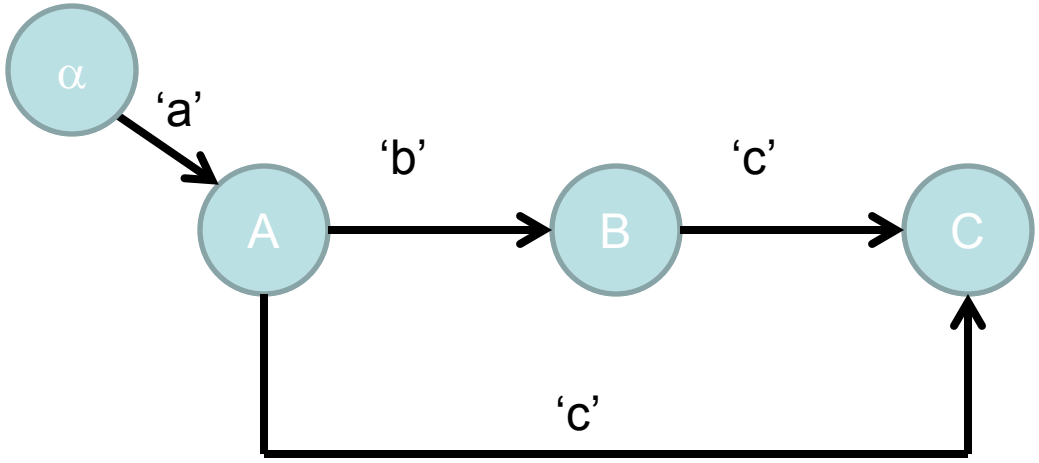
i = initial state

T = terminal state(s)

$\sum$ = alphabet of events

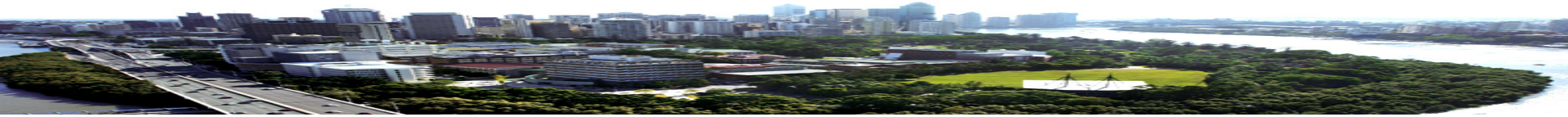
$\delta : S \times \sum \rightarrow S$

is the transition function



$S = \{\alpha, A, B, C\}$ $i = \alpha$ $T = C$ $\sum = 'a', 'b', 'c', other$

	'a'	'b'	'*c*'	***other***
α	***A***	α	α	α
δ: ***A***	***A***	***B***	***C***	***A***
B	***B***	***B***	***C***	***B***
C	C	C	C	C



Finite State Machines (Mealy)

$M = \{S, i, T, \Sigma, \Gamma, \delta\}$

S = set of states

i = initial state

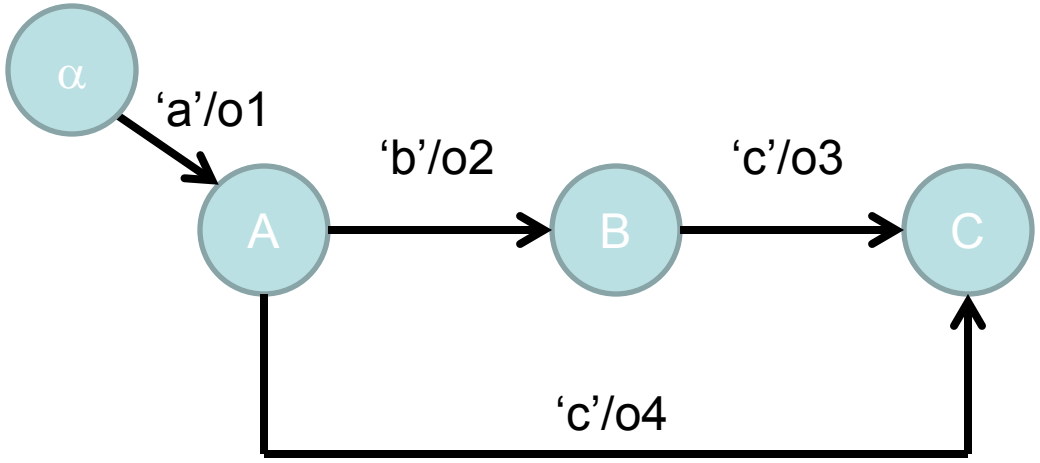
T = terminal state(s)

Σ = alphabet of events

Γ = set of outputs

$\delta : S \times \Sigma \rightarrow S$

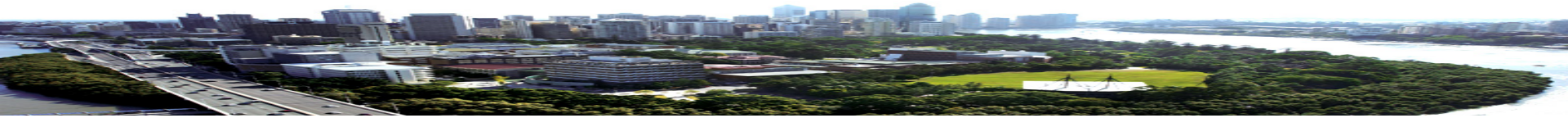
is the transition function



$S = \{\alpha, A, B, C\}$ $\quad i = S$ $\quad T = C$ $\quad \Sigma = 'a', 'b', 'c', other$

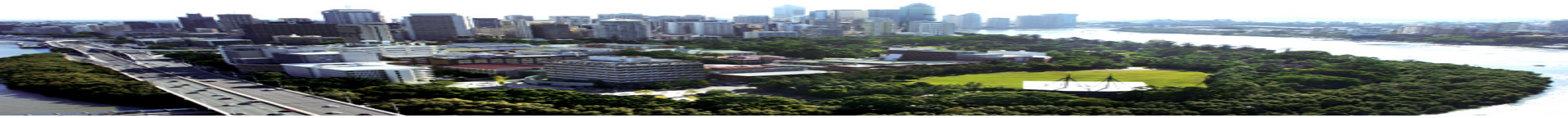
$\Gamma = \{o1, o2, o3, o4\}$ $\quad \delta :$

	'a'	'b'	'*c*'	***other***
α	***A*** / ***o***1	α	α	α
A	***A***	***B*** / ***o***2	***C*** / ***o***4	***A***
B	***B***	***B***	***C*** / ***o***3	***B***
C	C	C	C	C



Exercise

- Draw state diagrams for a system which takes strings of characters as input and recognizes the strings ‘ash’ and ‘cat’, respectively, and outputs ‘1’ and ‘2’, respectively.
- Can they run together within the same program?

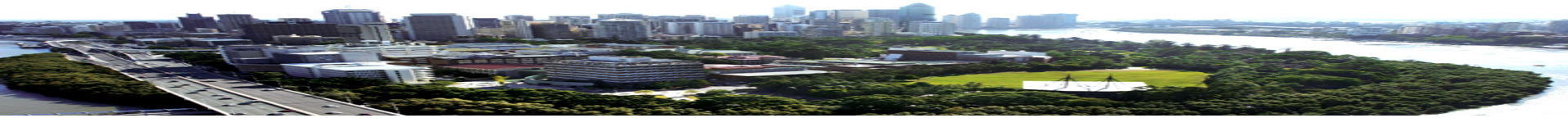


FSMs

- Easy to develop code for event driven systems
- Can be formally optimized
- Concurrency can be depicted using multiple machines

- Cannot describe ‘insideness’ or structure of modules
- Does not show how to break down into subroutines
- Communication for multiple FSMs can be difficult to depict
- Number of states can grow large and cause unmanageable complexity

State charts solve these problems



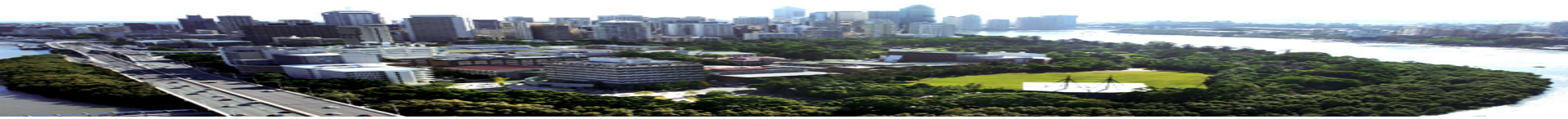
State charts

'FSM' + 'depth' + 'concurrency' + 'broadcast communication'.

Depth = state diagrams inside states possible. This facilitates modular development.

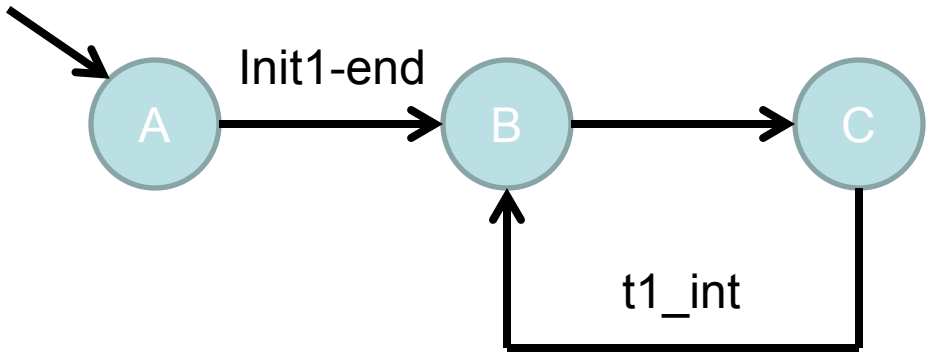
Concurrency = being in two states at the same time , represented with dashed lines separating states (also called orthogonal states or AND states). Concurrent processes can communicate with global memory if needed.

Broadcast communication = concurrent processes can react to the same event and thus be synchronized if necessary

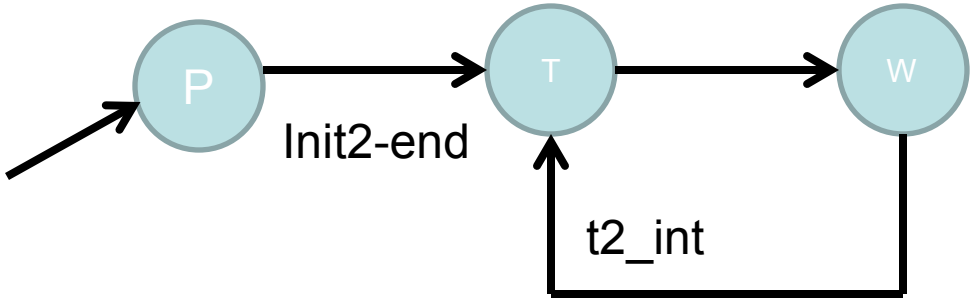


Concurrent processes

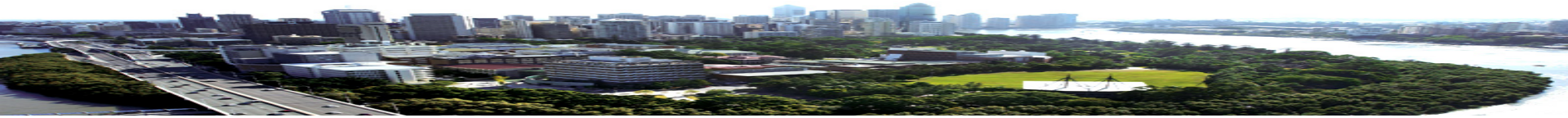
Q. How will the program switch from one diagram to another?



A = initialize
B = read and display
C = wait
t1_int = Timer 1 interrupt

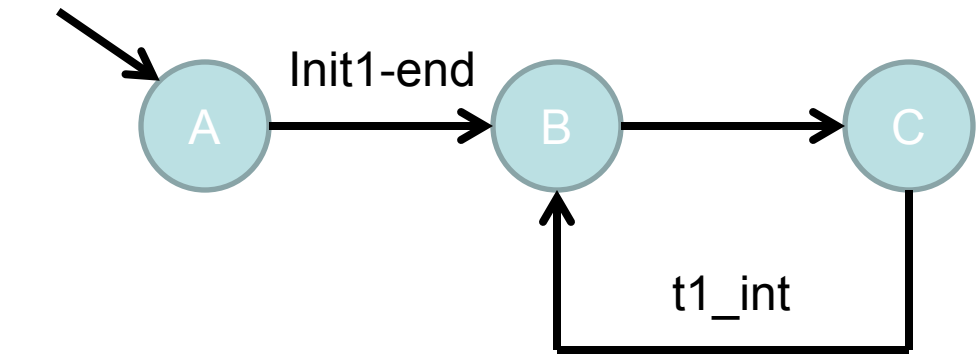


P = initialize
T = toggle LED on or off
W = wait
t2_int = Timer 2 interrupt

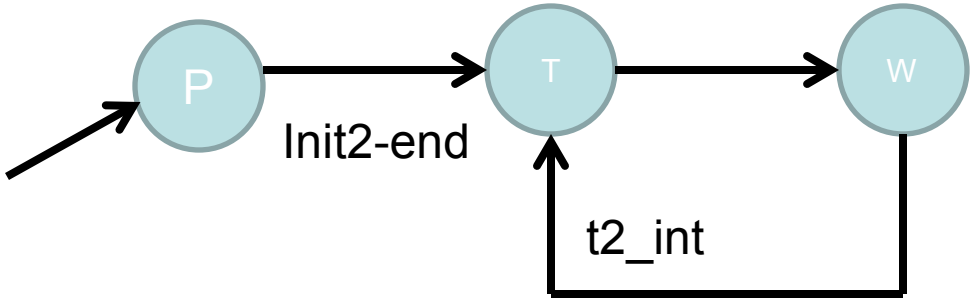


Concurrent processes

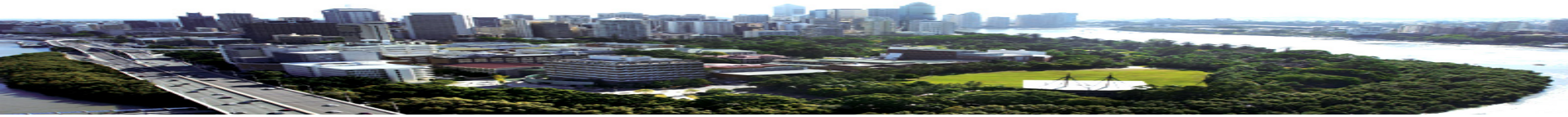
A. Either when the interrupts are processed OR at explicitly introduced kernel control statements like ‘yield’ or ‘waitfor’.



A = initialize
B = read and display
C = wait
t1_int = Timer 1 interrupt



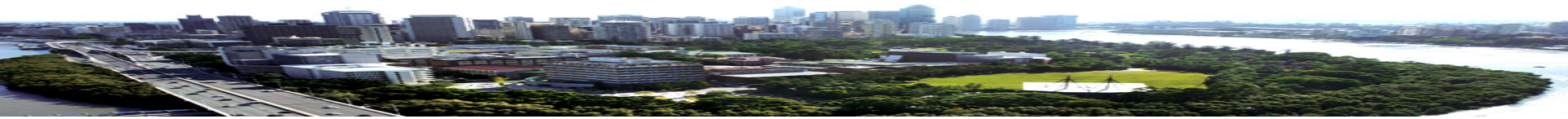
P = initialize
T = toggle LED on or off
W = wait
t2_int = Timer 2 interrupt



Exercise

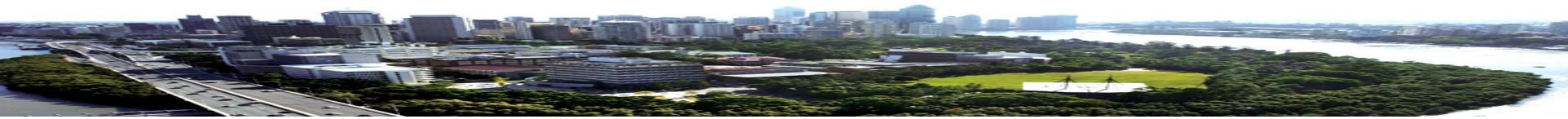
Draw a state chart representation for Task C of Laboratory exercise 1.

(It toggled an LED on/off every second and displayed the time every 10 seconds)

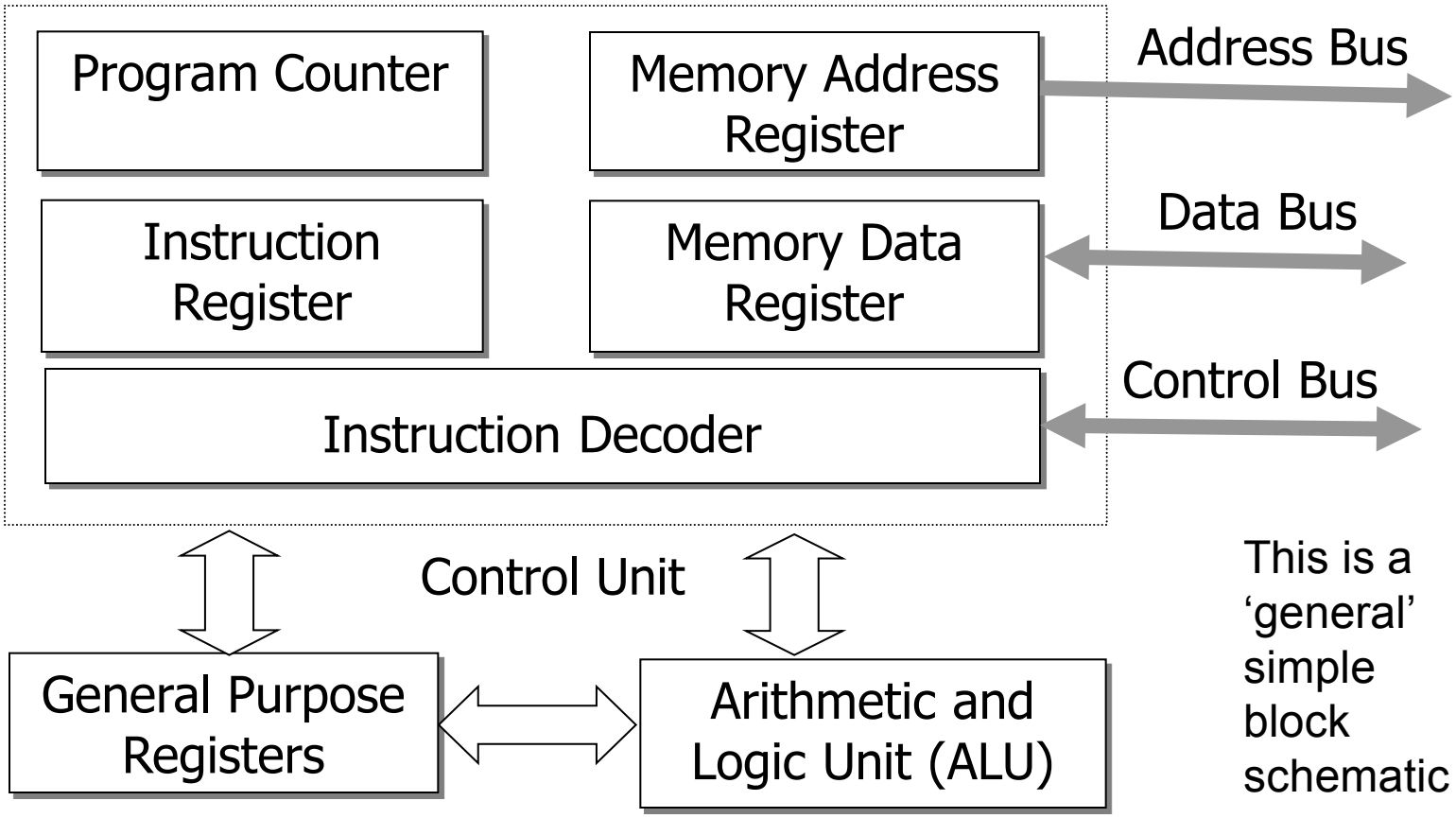


Demonstration

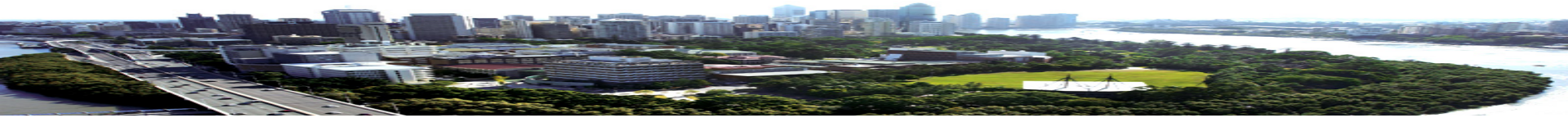
- One of the laboratory exercise 1 programs
- Using costates



CPU – simple schematic

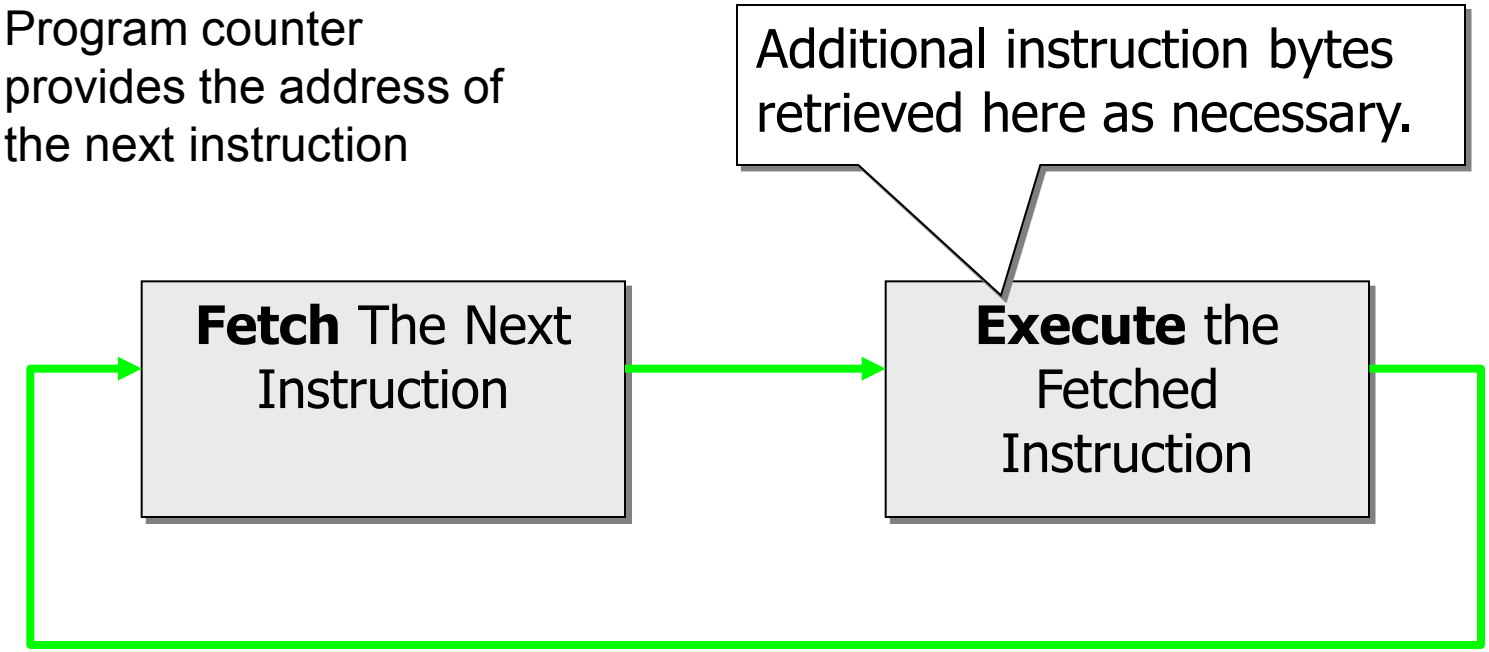


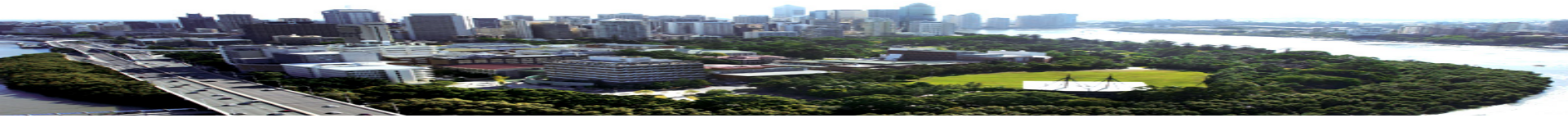
This is a 'general' simple block schematic



Fetch-Execute control cycle

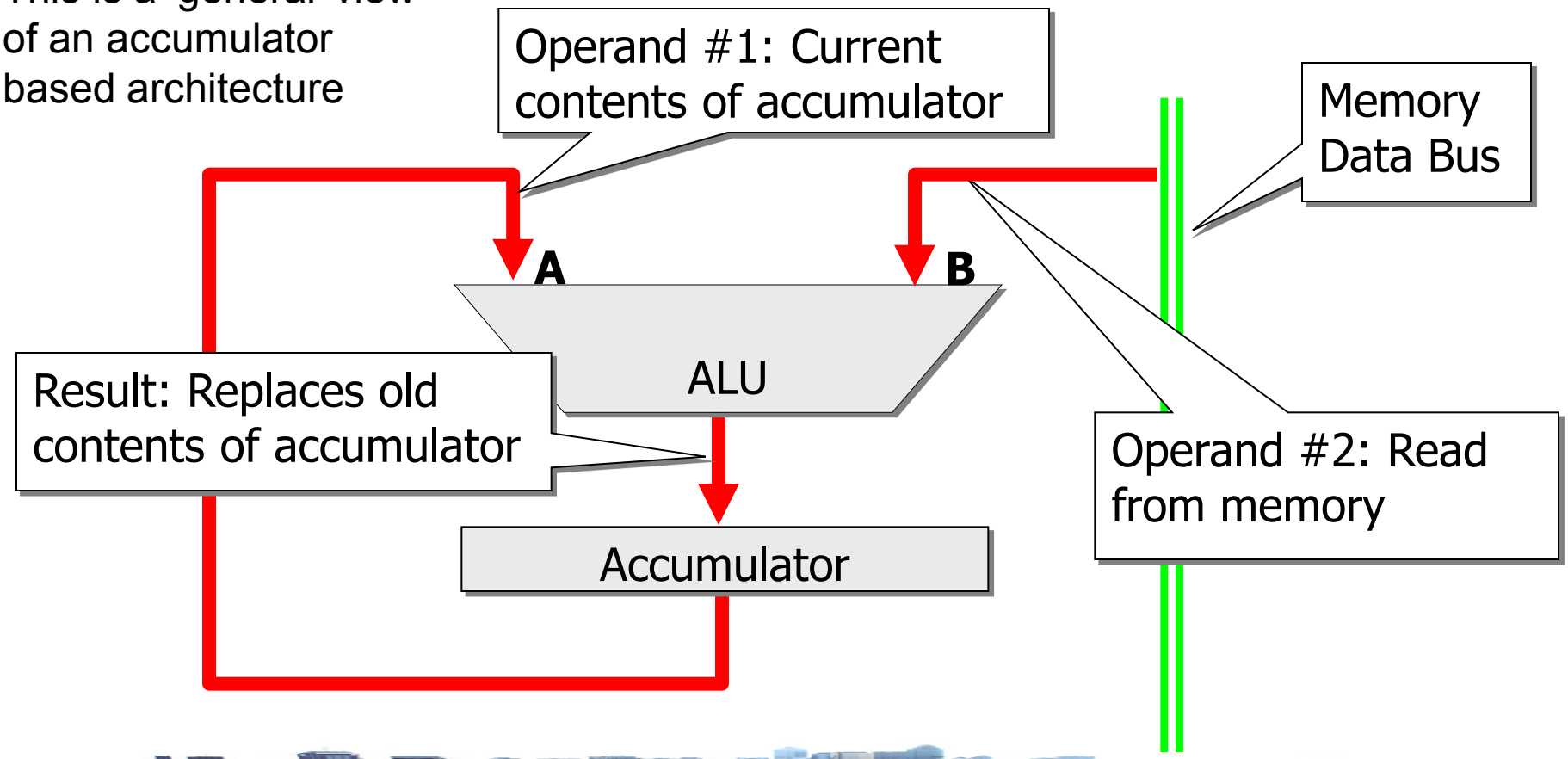
Program counter provides the address of the next instruction





ALU data paths for dyadic operations

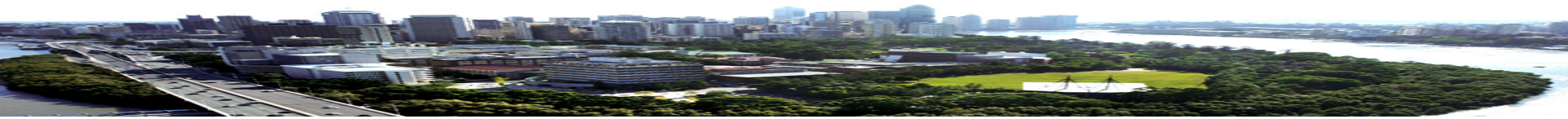
This is a 'general' view of an accumulator based architecture



Questions

- Where are the instructions and the data stored?

- Can there be more than one accumulator?



R4000 registers

To note:

8 bit, 16 bit and 32 bit possible

Extended program counter to extend memory

8 and 16 bit accumulators

2.2 Rabbit 4000 Processor Registers

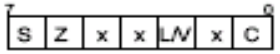
The Rabbit 4000 has a expanded register set over the previous two versions of the Rabbit chip.

Table 2. Rabbit 4000 Register Set

Registers	8-Bit	16-Bit	32-Bit	Alternate Registers
Accumulators	A	HL		A′, HL′
Flags[a]	F			F′
General Purpose	B, C, D, E, H, L	BC, DE, JK	BCDE, JKHL	B′, C′, D′, E′, H′, L′ BC′, DE′, JK′ BCDE′, JKHL′
Index		IX, IY	PW, PX, PY, PZ	PW′, PX′, PY′, PZ′
Stack Pointer		SP		None
Program Counter		PC		None
Xmem Program Counter	XPC	XPC (low 12 bits valid)		None
Interrupt Priority	IP			None
Internal Interrupt	IIR	SP		None
External Interrupt	EIR	PC		None
System/User Mode	SU			None
Handle Table Register	HTR			None

a. S=Sign, Z=Zero, LV=Logical/Overflow, C=Carry. Bits marked "x" are reserved for future use.

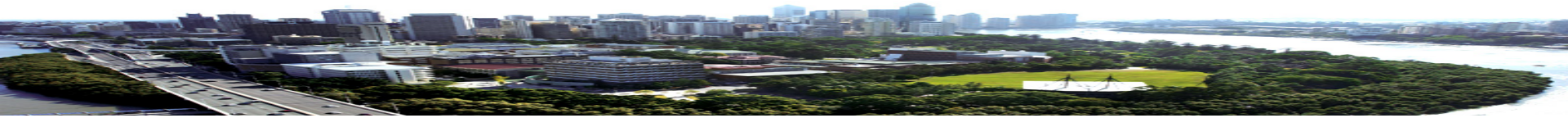
Flag Bits



On page 22 of Rabbit 4000 Instructions manual

Other documents: R4000 user manual

R4000 Designers handbook

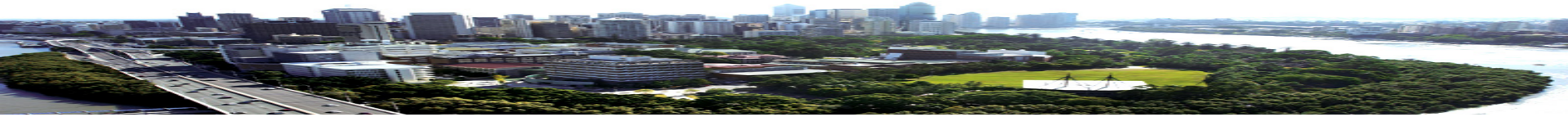


Exercise

Using the microcontroller architecture poster and documents identify and answer:

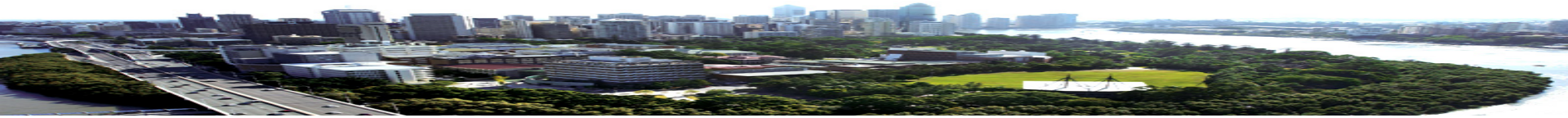
- What is the address bus size?
- How much memory can the system address?
- What are the segments in the logical address map?
- What are some of the registers associated with parallel ports?
- What are some of the on-chip peripherals?
- Does it have a real-time clock on the chip?
- Does the R4000 prioritize external interrupts?

References: R4000 user manual, R4000 designers handbook



Real-time clock

- Keeps track of time
- Can keep time in secs, minutes, hours etc and perform calendar functions
- Timer runs free of system clock
- Registers in the RTC can be read by the application program running on the microcontroller
- RTC can be based pm software based timing loops, external chip or internal peripheral
- Rabbit4000 has built-in RTC



What is a real-time clock?

- What is a real-time clock (RTC)?

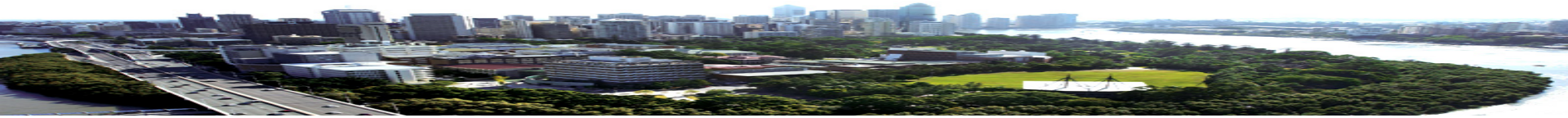
  External timer and registers that keep time-of-day

- Where is it on the system?

  Peripheral on the chip or outside

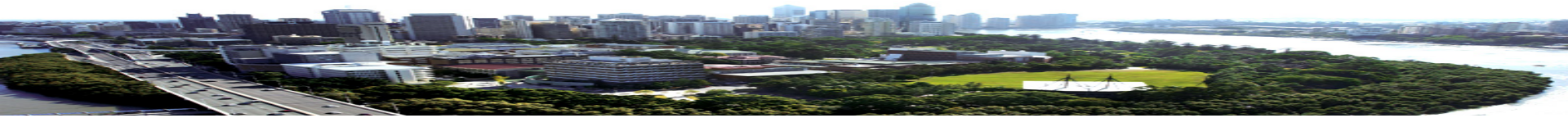
- What happens to the RTC when the system power is off?

  It is battery backed and does not lose data



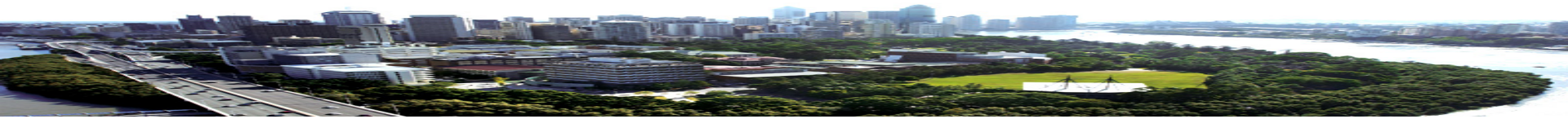
Using the Rabbit RTC

- Rabbit4000 on-chip RTC has 6 clock byte registers, global control/status register, RTC status register
- Library functions allow reading time, conversion etc.
- unsigned long int read_rtc(void) - reads seconds
- void write_rtc(insigned long int time) – writes seconds
- structure tm (standard C structure) to communicate calendar information (tm_sec, tm_min, tm_hour, tm_mday, tm_mon, tm_year and tm_wday). It can be read or written using tm_rd and tm_wr.
- See sample program used in laboratory exercise 1



The prototyping board

- Rabbit Core Module RCM4000
- Has the R4000, ethernet controller, A/D chip, switches, LEDS etc.
- LCD/keypad has been interfaced with it via a serial port connection
- There is a particular configuration of the R4000 ports for the board



RCM4000 features

RC4000UM.pdf

On page 2

Table 1. RCM4000 Features

Feature	RCM4000	RCM4010
Microprocessor	Rabbit® 4000 at 58.98 MHz	
SRAM	512K	
Flash Memory (program)	512K	
Flash Memory (mass data storage)	32 Mbytes (NAND flash)	—
A/D Converter	12 bits	—
Serial Ports	5 shared high-speed, CMOS-compatible ports: 5 are configurable as asynchronous serial ports; 4 are configurable as clocked serial ports (SPI); 1 is configurable as an SDLC/HDLC serial port; 1 asynchronous serial port is used during programming 1 asynchronous serial port is dedicated for A/D converter (RCM4000)	

Rabbit 4000 ports in RCM4000

Note:
Serial ports C and D are using Port C pins;

Pins can be configured in alternate ways

Figure 7 shows the use of the Rabbit 4000 microprocessor ports in the RCM4000 modules.

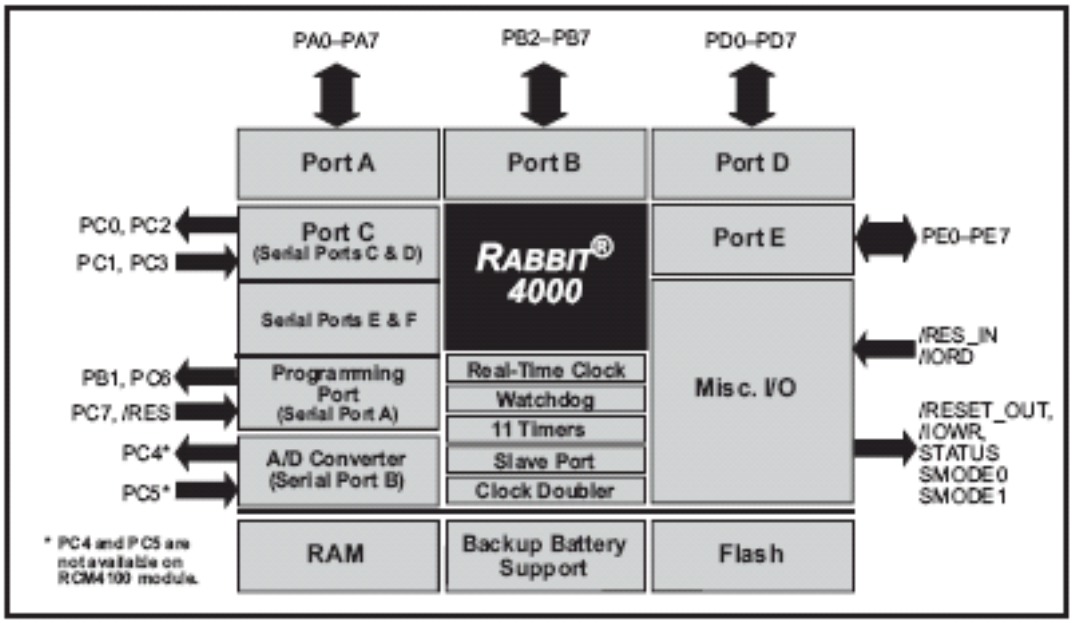
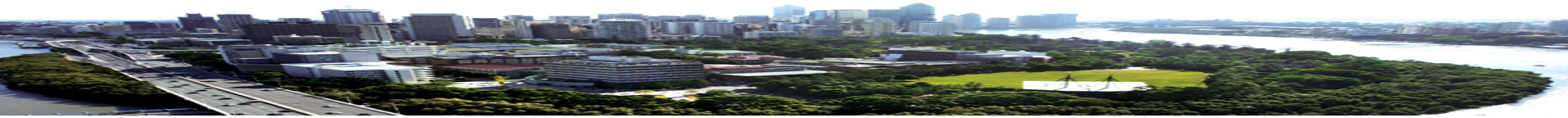


Figure 7. Use of Rabbit 4000 Ports

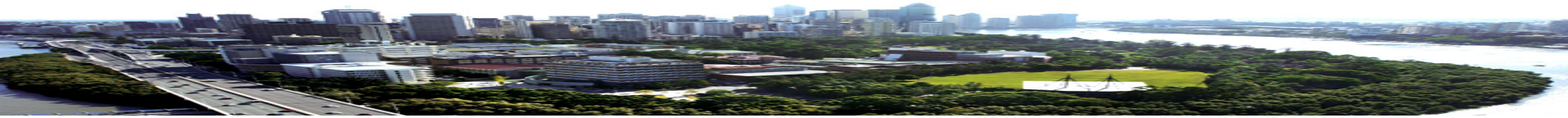
Parallel port B is used to map the LEDs and switches.

The Festo interface uses parallel port A and B pins.



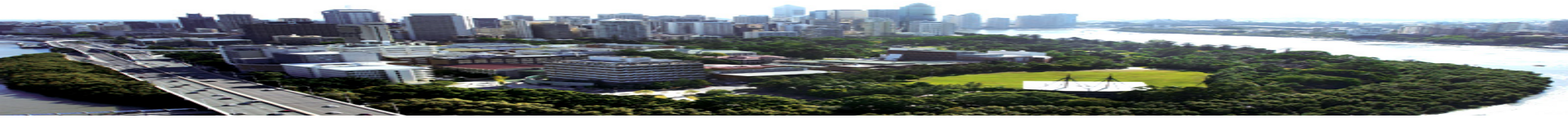
Serial port usage

- Lab hardware (box enclosure with RCM4000) provides
  - RS232 DB-9 connection to serial port D OR LCD/keypad connected to it (switch)
  - RS232 DB-9 connection to serial port C OR Zigbee module connected to it (switch)
  - A DB-25 connector at the back for Festo testing station interface which uses parallel port A and B pins (other than PB0,PB1)
  - Serial port A is the programming port for the RCM4000
  - Serial port B is used by the A/D convertor
  - Note that serial ports A,B,C,D use parallel port C pins. PB0 and PB1 are used by them for clocking.

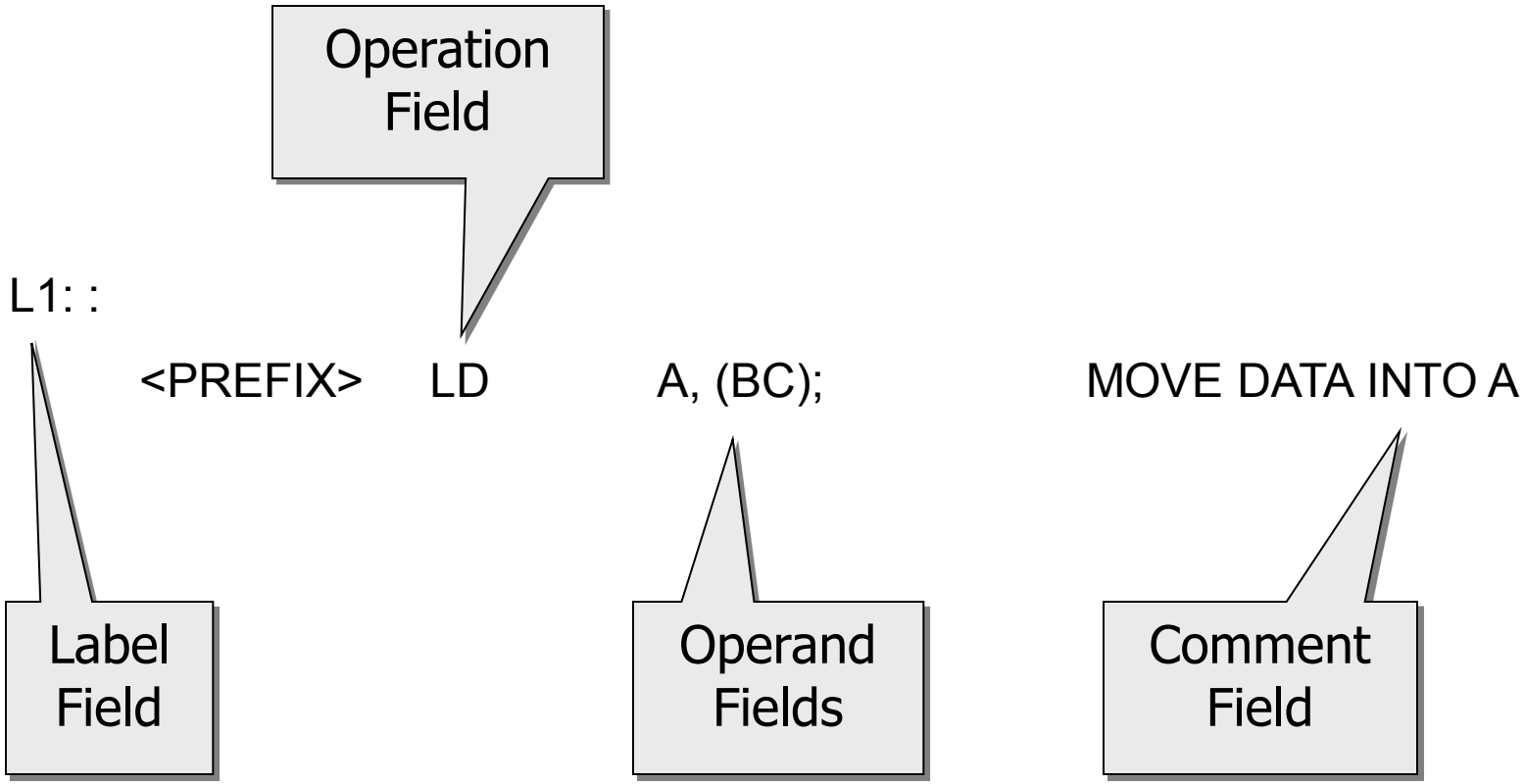


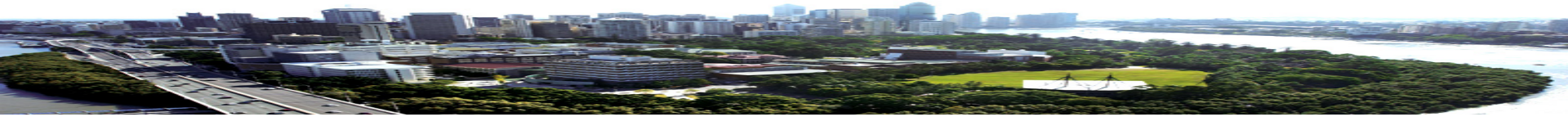
Instruction set

- Instructions are provided with a microcontroller to manipulate data in registers in desired ways – logic, arithmetic, data movement etc.
- The instruction set is easier for human reading in assembly format.
- Compilers translate from a high level language to assembly code



Rabbit Dynamic C - Assembler format

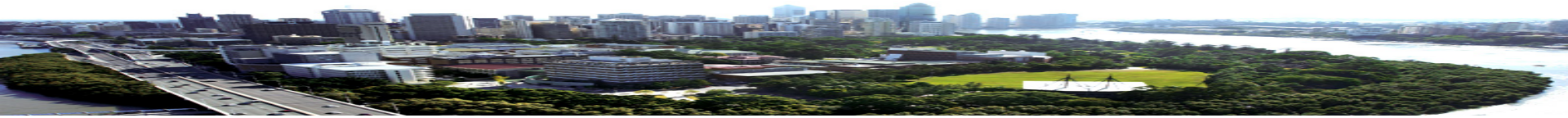




Instruction format and groups

- By operand number
  - Zero, eg MUL
  - one, eg INC IX
  - Two eg ADD JK,HL
- By operand type
  - Constant
  - Register
  - Memory location (16 bit address normally, 24 bit address access by special instructions on R 4000)
  - i/o port
- By operand size
  - 8 bit
  - 16 bit
  - 32 bit
- By type of operation
  - Arithmetic logic bit shift/rotate
  - Data movement : block copy stack
  - Control transfer supervisor/user mode interrupt encryption support address translation

Rabbit 4000 instructions available in the document RabbitInstructions.pdf



R4000 instruction (example 1)

Add with carry (CF)

8 bit operation, result in A

Memory Address (16 bit) in HL

ADC A,(HL)

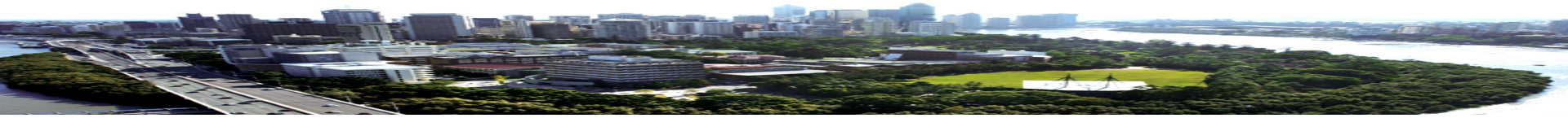
Opcode	Instruction	Clocks	Operation
7F 8E	ADC A,(HL)	7 (2,2,1,2)	A = A + (HL) + CF

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•		•	

Description

The data in A is summed with the C flag and with the data whose address is in HL. The result is stored in A. The Rabbit 4000 assembler views "ADC A,(HL)" and "ADC (HL)" as equivalent instructions. In the latter case, A is used even though it is not explicitly stated.

The opcode for this instruction is different than the same instruction in the Rabbit 2000, 3000 and 3000A.



R4000 instruction (example 2)

32 bit operation

Bitwise AND

Bitwise AND 4000

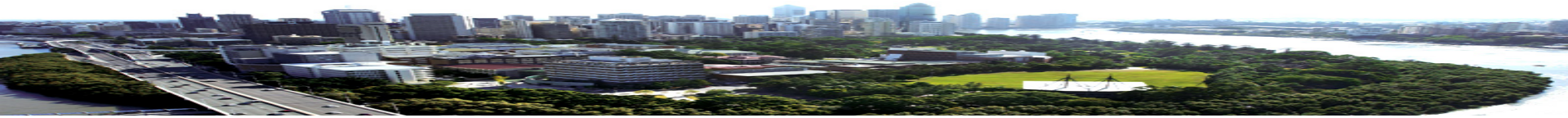
```
AND JKHL,BCDE
```

Opcode	Instruction	Clocks	Operation
ED E6	AND JKHL,BCDE	4 (2,2)	JKHL = JKHL & BCDE

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

Performs a bitwise AND operation between the 32-bit registers JKHL and BCDE. The result is stored in JKHL.



R4000 instruction (example 3)

8 bit operation, loads A

With contents of

Memory Address (16 bit) in register pair BC or DE or immediate constant

Load 2000, 3000, 4000

```
LD A,(BC)
LD A,(DE)
LD A,(mn)
```

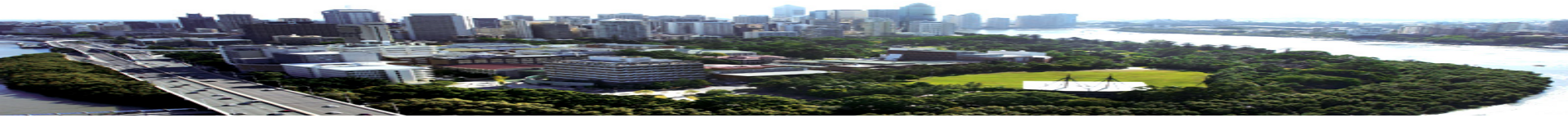
Opcode	Instruction	Clocks	Operation
0A	LD A,(BC)	6 (2,2,2)	A = (BC)
1A	LD A,(DE)	6 (2,2,2)	A = (DE)
3A *n m*	LD A,(*mn*)	9 (2,2,2,1,2)	A = (*mn*)

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•		•	

Description

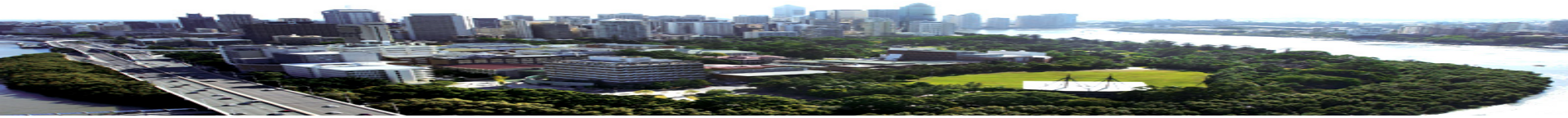
Loads A with the data whose address is:

- BC, or
- DE, or
- the 16-bit constant *mn*.



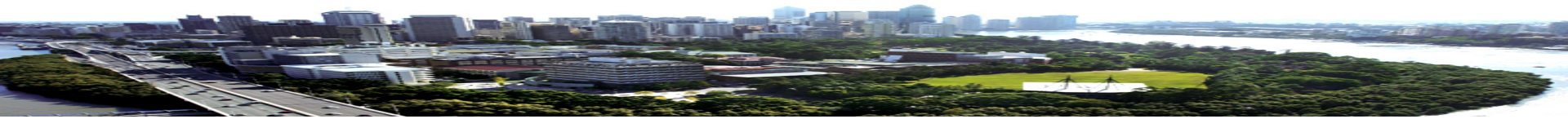
Arithmetic instructions

- 8 bit, 16 bit and 32 bit operations
- Add, Add with carry, Subtract, Subtract with borrow
- 16 bit Multiply unsigned (MULU) and signed (MUL) with result in 32 bit register
- No divide, No floating point co processor
- Dynamic C uses library functions for real arithmetic
- Can represent real numbers as fixed point and write library of procedures to support real arithmetic



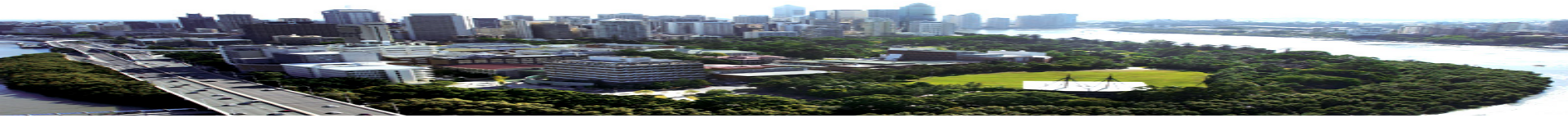
Stack

- Special region of memory in which data is ordered
- Data stored in same order as written; retrieved in reverse order
- LIFO (Last In First Out)
- Stack Pointer – holds address of stack top
- PUSH and POP instructions
- Stack used extensively by procedure calls to store return address, parameters and local variables
- In assembly code referred to sp refers to the stack pointer and (sp) the contents of at sp.



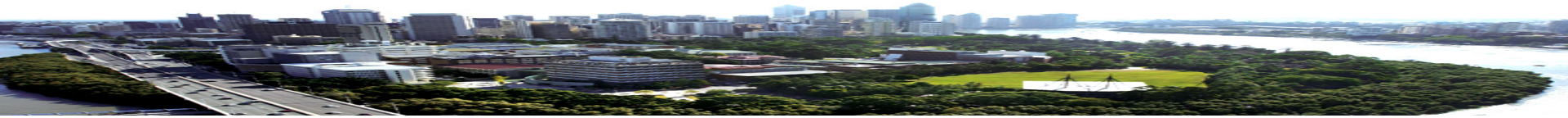
Instructions summary

- Little-endian - Numbers are stored low-byte first.
- Signed and Unsigned numbers
- 2s Complement representation.
- 8 bit accumulator A, 16 bit accumulator HL
- BCDE and JHKL are 32 bit registers
- PC,SP are 16 bit



Demonstration program

- To see register values
- To watch variables
- To see disassembled code
- To single step code
- To find out the difference between global variables and those inside a C function



Watching variables and register values

- The following step can be useful while debugging in Dynamic C
  - Open the program
  - Edit as necessary and compile the program
  - Click ‘Inspect’. Click ‘Add watch’. Enter the name of a program variable you want to examine as you step through code
  - Click ‘Inspect’. Click ‘Disassemble at Cursor’ to view assembly code. When you are in this window you can single step assembly.
  - Click ‘Window’. Click ‘Debug Windows’ and click ‘Registers’ to open the window and watch registers.
  - You can cascade and arrange multiple windows to suit
  - Press F7 to single step