J.d.

* Hard core:

A hard processor core has dedicated silicon on the FPGA. This allows it to operate with a core frequency.

A benefit it provides is that it exists in an environment where the surrounding peripherals can be customized for the applicat

* Soft core:

A soft core processor is one that is implemented entirely in the logic primitives of an FPGA. And best of this implementation it will not operate at the speeds like hard core

It's appropriate for a simple systems like GIPO (Gen purpose I/O). Also fits in a complex system if an OS is incorporated

06.03.14.
THU.

AFTER MIDSEM

" IC Technology & Design Technology."

* Car
* Cell Phone
* Aeroplane
* Steel plant

1. Single IC or multiple.
2. System on chip.

Top down design
Bottom-up design → a few analog ICs. (very simple digital) designs
Mixed mode (Top down & Bottom up)

IC Design:

1. Full custom Design
2. Semi custom Design
   a. Standard cell.
   b. Gate array.
3. FPGA based design CPLD based designs.

[Concerns. Time & Money.]

Design: Structural → Layout (Physical design) → Placement, Routing, Padding. → Masks. → Layering → IC Fabrication → Testing.

ALU

alusel 2	alusel 1	alusel 0	operation name	operation name	Instruction.
0	0	0	Pass	Pass A to o/p	non ALU Instruc.
0	0	1	AND	A AND B	AND A, rrr
0	1	0	OR	A OR B	OR A, rrr
0	1	1	A'	NOT A	NOT A
1	0	0	A+B		ADD A, rrr
1	0	1	A-B	A-B	SUB, A, rrr
1	1	0	Increment	A+1	INC A.
1	1	1	Decrement	A-1	DEC A

Shifter:

sel 1	sel 0	Instruction.
0	0	Pass.
0	1	SHFLA.
1	0	SHFR A.
1	1	ROTR A.

THU.
20-03-2014

Control Word:

Number eg: ADD A, 011

muxed 1	muxed 0	accwr	rst	rfwr	rfadr$_2$	rfaddr$_2$	rfaddr$_0$	alusel$_2$	alusel$_1$
0	0		0	0	0	1	1	1	0

alusel 0	shiftersel 1	shiftersel 0	outer.
0	0	0	0

eg. ADD A, 100

0 1 1 0 0 1 0 0 0 0

0 1 0 0

eg. MOV 100, A.

0 1 1 0 1 0 1 0 0 0 0

0 1 0 0

Fri
21.03.2014

Instrumentation Set:

1. Data movement ⑤
2. Jump instruction. ⑧
3. ALU " ⑩
4. In/out instruction ②

25

Instruction Encoding.

Data movement instruction. Operation/Comment.

Load accumulator/Transfer

LDA A, m — 0001 | 0m

STA m, A — 0010 | 0m — Load register from accumulator

LDM aaaaaaa, A — 00110000 00aaaaaa — Load memory " "

STM aaaaaa, A

LDI A, iiiiiiii — 0100000 00aaaaaaa

PC = aaaaaa

if (A == 0) then PC = aaaaaa

if (A != 0) then PC = aaaaaa

if (A == +ve) then PC = aaaaaa

if A = +ve & and smmm = 0 the n.

if s = 0 then PC = PC + mmm

ALU Instruction:

AND	A, m	1010 0m
OR	A, m	1011 0m
ADD	A, m	11000 m
SUB	A, m	11010 m
NOT	A	1110 000
INC	A	1110 0001
DEC	A	1110 0010
SHFL	A	1110 0011
SHFR	A	1110 0100
RCTR	A	1110 0101

Monday.
24.03.2014.

Control Unit:

Design fsm for the CU which will basically cycle through -

four main states:

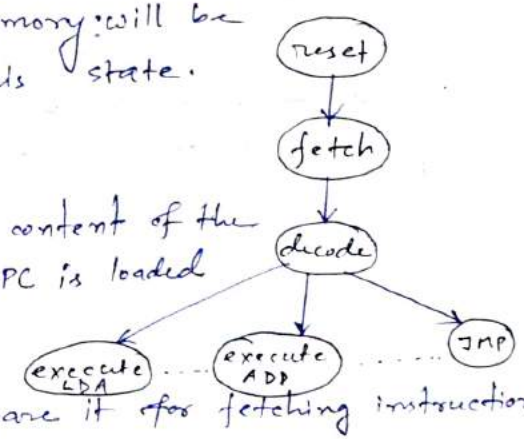
i) reset
ii) fetch
iii) Decode &
iv) Execute.

i) Reset: → fsm will start from reset state.

→ when in reset state, it will initialize all the control signals & working variables.

variables: PC - Prog. counter.
IR - Instruction register
state - state variable.

→ content of memory will be loaded in this state.



ii) Fetch:

a) The memory content of the location pointing to PC is loaded in the IR.

→ PC = PC + 1 → to prepare it for fetching instruction.

→ JMP → (PC)

iii) Decode:

→ The content that is stored in the IR is decoded according to the encoding that is assigned to the instructions.

→ Case (opcode)

→ Look up table → encoding.

→ execution state.

iv) Execution:

→ It sets up the control word which asserts the datapath to carry out the execution.

→ Each instruction has its own execute state.

→ Control word.

→ At the end of execute state, FSM goes back to fetch state & the cycle repeates for the next instruction.

Thu. 27.03.14.

ARM (Acorn RISC Machine) :- England.

ARM1 → 25,000 trans → 6MHz.

ARM2 → 30,000 trans
32 bit data bus.
26 bit address bus.
16 no. & 32 bit register.

#Advanced Features: (Book: ARM Developer guide)

[i] Thumb:- (32 bit processor) It works in 16 bit only.

[ii] MMU | MPU | CASH :

[iii] Debug Interface.

[iv] ICE (In Circuit Emulator)

[v] Fast multiplier.

[vi] Enhanced DSP Instructions

[vii] Ja Zelle data byte operation - 8 bit.
(JAVA)

[viii] Floating point operation.

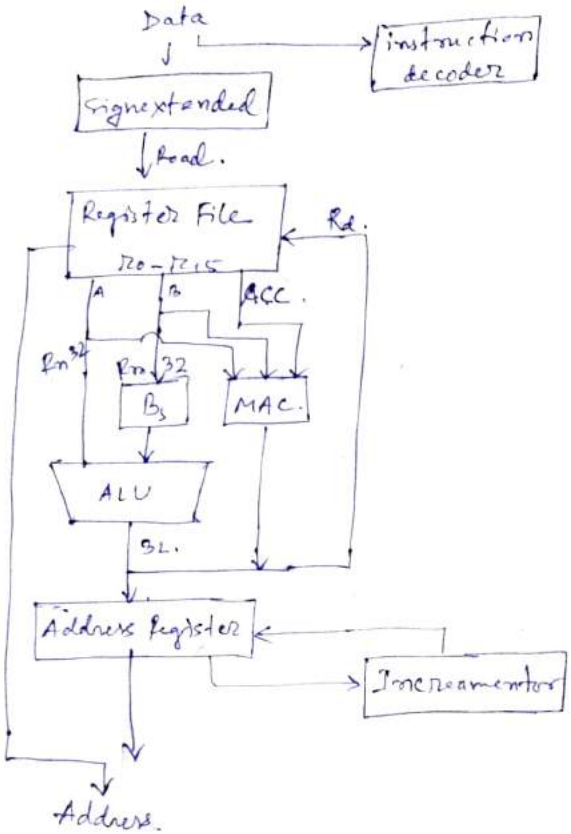
[ix] Synthesizable :- RTL code with licenses extensions & modifications are possible.

Nomenclature of ARM:

Synthesis-able

ARM {x}{y}{z}{T}{D}{M}{I}{E}{J}{F}{S}

- x → Family. 7, 8-9_10.
- y → MMU / MPU
- z → cache.
- T → Thumb
- D → Debug interface.
- M → Multiplier
- I → ICE
- E → Enhanced DSP Instruction
- J → JAVA Ja Zelle
- F → Floating Point instr
- S → Synthesis-able

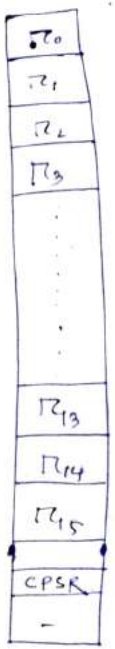


[Basic Architecture of ARM Processors]

Fri
28.03.14.

Fri
28.03.14.

Registers in ARM core



i) Active Registers are of 32 bit size.

ii) 18 active registers :- 16 data registers, 2 processor status register.

iii) 3 register assigned to a particular task.

r_{13} → stack pointer – stores the head of the stack.

r_{14} → link register – the core puts the return address whenever it calls a subroutine.

r_{15} ⇒ programming counter – contains the address of the next instruction to be fetched by the processor.

∴ 2 program status register.

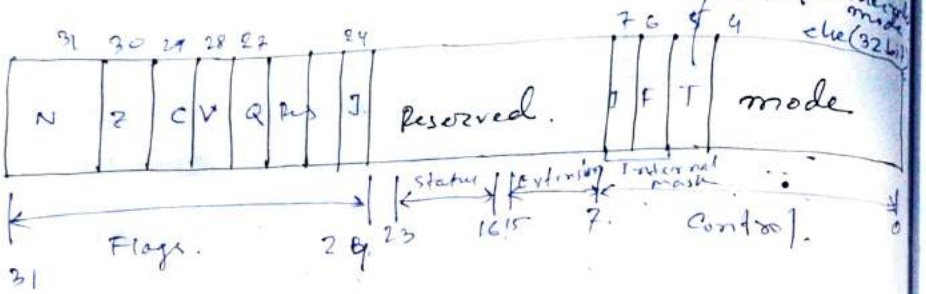
(Current programmed status register)

(Saved " " ")

Current ~~status~~ Program status Register:

→ use cpsr to monitor & control internal operations.

→ 32 bit register.



6 - F F = 1 disable FIQ interrupts (Fast Interrupt Request)

F - I I = 1 disables IRQ interrupts (Interrupt Request)

=> I = 1 have execution instruction execution.

Q = saturation Flag. (QADD)

C = Overflow flag → unsigned overflow for addition.

V = overflow flag :- signed additions.

Z = zero flags if the result is zero

N = Negative. flag → 31 = MSB

± Saved Program Status Register :-

1) For exceptions and interruptions.

2) When an exception occurs, the corresponding SPSR saves the CPSR value into it.)

So as to retrive it on returning to the previous mode.

3.) User mode does not have SPSR.

Processor mode :-

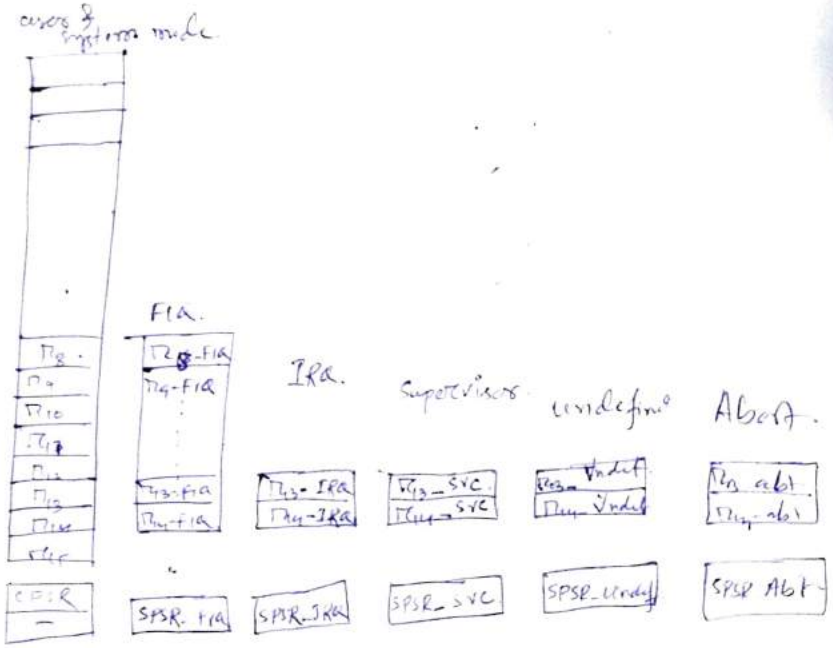
Privillege mode — full read/write access of the CPSR,

non-privilleged mode - Read access to the control field and read/write of the flag field.

→ 6 → Abort → when failed attempt to access memory
→ FIQ → High priority interrupts
→ IRQ → Low " "
→ Supervisor → Processor is after reset & O.S kernal operates.
→ System → allows full R/W of CPSR.
→ undefined → processor encounters an instruction that is undefind or not supported.

Banked Registers in ARM :- Core || Processor modes

16 registers — GP
= CPSR
+1
→ 20 Reg → reg are dedicated for modes.



Eg. interrupt request mode.

If mode is changed, a banked register from new mode will replace by an existing register.

In I instructions → r_{13} - irq, r_{14} - irq.

What happens when an interrupts request forces a mode change?

i) registers changes to banked registers.

ii) It contains the stack pointer for IRQ - r_{13} - IRQ

iii) It " " return address for IRQ. r_{14} - IRQ

iv) A new register in this mode is program status register. Stores the previous modes CPSR

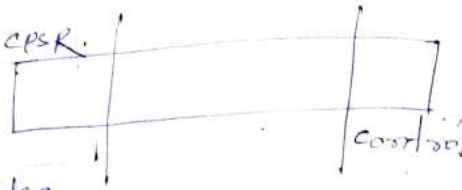
v) To return back to user mode, a special return instruction is used that restore original CPSR from the SPSR-IRQ & banked register in the r_{13} & r_{14}.

Mon
31.03.2014

ARM Core:

→ 37 registers (20+16+1) →SPSR.

Mode	Abbreviation	Previlleged	Mode[4:0]
Abort	abt	yes	10111
FIQ	FIQ	yes	10001
IRQ	IRQ	yes	10010
Supervisor	svc	yes	10011
System	sys	yes	11111
Undefined	Und	yes	11011
User	user	~~yes~~ no	10000



(n/z/ → small ↓ 0)

C=1 is set to 1.

⇒ 32-ARM Core.

State Instructions:

ARM	THUMB	Jazelle.
32-bit	16-bit	8 bit.
	1	0
55 no instrct.	30 no instrch	

Exceptions, interrupts & the vector table.

Vector table:

Exception/Interrupt	Shorthand	Address	Mem. Loc^n
1. Reset (Power is applied) initialise	RESET	0X00000000	
2. Undefined Instruction (processor wont decode an instruction)	UNDEF	0X00000004	
3. Software interrupt. (SWI → instruction.) invoking os.	SWI	0X00000008.	

4. Prefetch abort. PABT 0X0000000C.
(processor attempts to fetch an instruction from an address without correct permission)

5. Data abort.

⇒ # 02.04.14 & 3.4.14 lectures are at the last.

Monday
07.04.2014

Multiple Register Transfer.

→ load store multiple instructions can transfer multiple registers betn memory and the processor in a single instruction

→ base address. (Rn) → point to a memory location

→ stacks.

syntax:- <LDM|STM>{cond}<addressingmode>
Rn{!}, <registers>{^}

LDM - Load multiple registers.

STM - Store " "

Addressing mode	Description	start addr	End addr	Rn
IA	Increment after	Rn	(Rn+4*N-4)	Rn+4*N
IB	" before	Rn+4	(Rn+4*N)	Rn-4+4*N
DA	Decrement after	Rn-4N+4	(Rn)	Rn-4*N
DB	" before	Rn-4N.	Rn-4	Rn-4*N

LDM IA r0!, {r1-r3}

Pre: mem32 [0x80018] = 0x03
mem32 [0x80014] = 0x02.
mem32 [0x80010] = 0x01

r0 = 0x00080010.
r1 = 0x00000000
r2 = 0x00000000
r3 = 0x00000000

Port	
0x80018	0x00000004
0x80014	0x00000003
0x80010	0x00000002
0x8000C	0x00000001
0x80008	0x00000000

Wed 02.04.14

movs r0, r1, LSL#1
↳ update status

Pre
r0 = 0x00000000
r1 = 0x80000004
CPSR = nzCvqiFt_user
↳ carry high., others small (low)

After executing the instruction.

Post
r0 = 0x0000 0008.

Arithmetic Instructions:

ADC ADD RSB (Reverse subtract) RSC SBC SUB.

Rd = Rn + N + Carry.
↳ Rm / imm / RS LSL #1 → Logical shift left.

RSB: Rd = Rd - N - !C
RSC: Rd = N - Rs - !C

Pre
CPSR = nzCvqiFt_USER.
r1 = 0x0000 0001
SUBS r1, r2, #1.

Post
r1 = 0x00000000
CPSR = nZCvqiFt_USER

Logical Instructions:

Syntax: <instruction>{cond}{s} Rd, Rn, N

AND - logical bitwise OR
ORR - Rd = Rd | N (→ OR)
EOR = Rd = Rd ^ N
BIC = Rd = Rn & ~N

Comparision Instruction:

CMN - Compare Negative - Flags set as a result of Rn - (-N)

CMP - Compare Flags set as a result of Rn - N.

Multiply Instruction:

Syntax - MLA {cond} {s} Rd, Rm, Rs, Rn.

MUL {cond} {s} Rd, Rm, Rs.

MLA = Multiply & accumulate. Rd = Rm * Rs + Rn.

MUL = Multiply = Rd = Rm * Rs.

Long Multiplication:

SMLAL	Signed..	long [RdHi; RdLo] = RdH, RdLo + (Rm x Rs)
SMULL		
UMLAL	unsigned...	long
UMULL		

r_0 = 0X0000 0000

r_1 = 0 x 0000 0000

r_2 = 0 x F0000002

r_3 = 0 x 00000002

UMULL r_0, r_1, r_2, r_3

r_1 = 0 x00000001 = RdHi

r_0 = 0 x00000004 = RdLo

Branch Instruction:

B = Branch.

BL = Branch with link.

Bx = Branch exchange

BLx = Branch exchange with link.

Thu 03.04.14

LDR $r_0, [r_1, \#0]$. STR $r_0, [r_1]$

Modes of Addressing:

1. Pre-index with write back LDR $r_0, [r_1, \#4]!$
2. Pre-index LDR $r_0, [r_1, \#4]$.
3. Post-Index LDR $r_0, [r_1], \#4$

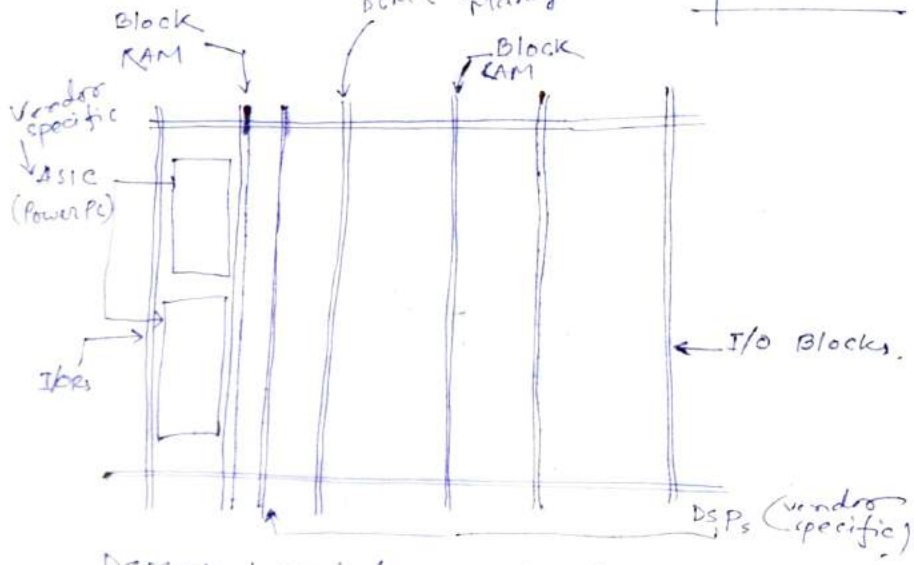
SMPS — Lab — DC
TV — 5V
Microwave — 4.9V or 5.0001V

* It has to be accurate.

with FPGA based design it may not be possible to ~~achieve~~ achieve the exact accuracy demanded by the customers.

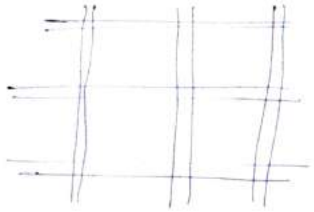
Pros: [i] Overnight Design.
[ii] FPGAs are very cheap.

07.03.2014
Fri



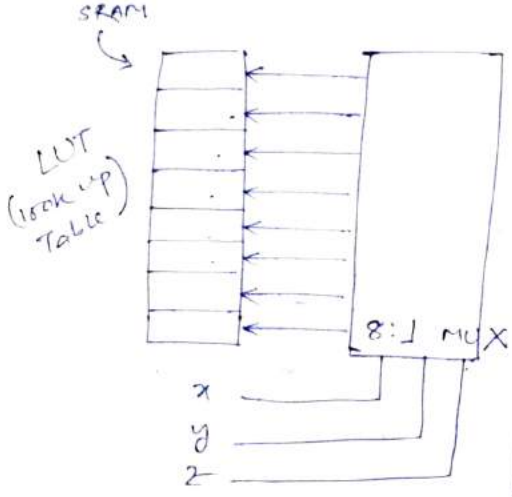
DCM → 1 crystal → single freq. but many PLLs.
PLL1
PLL2

CLB (Configurable Logic Block):

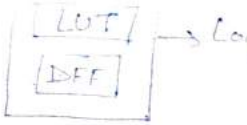


$f(x, y, z) = xy + z'$

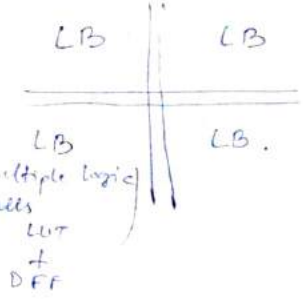
x	y	z	f
0	0	1	0
0	0	0	1



* using these look up table all the calculations are made. And this is done in a offline mode.

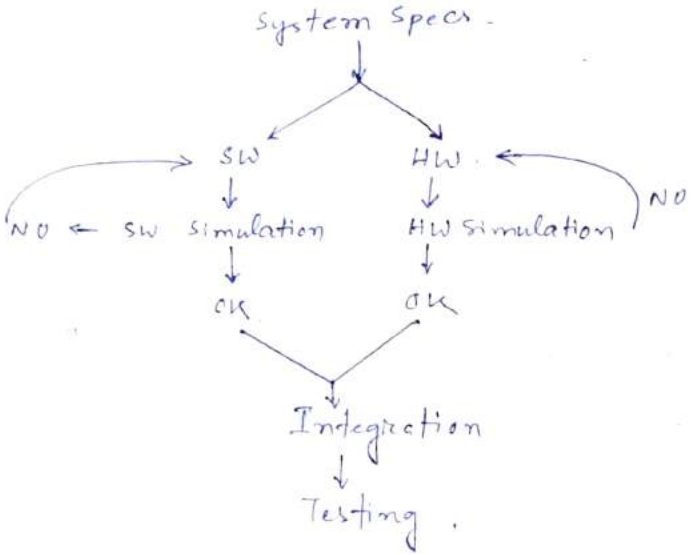


Slices of logic cells
multiple of logic cells
↓
LOGIC BLOCK.



Mon.
10-03-14

Hardware Software co-simulation:



Check points:

1. Check with the specs that are actually complied to the user requirements.

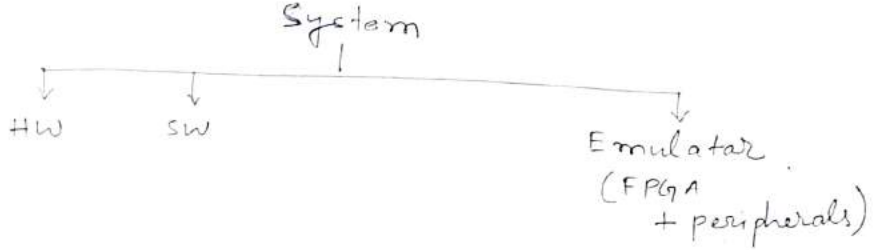
2. Check for simulation results of both HW & SW seperately.

3 Verification should be done after integration.

S/W simulation → C/C++.

HW " → HDL (Hardware Description Language)

#Emulator: Shows a characteristics that is very similar to that of the original device.



Low Power Design in Embedded systems:

1. Recharging.
2. Size (battery tech. has not scaled up as our VLSI tech.)
3. Power down mode.
4. I/O Devices.
   - Relay
5. Speed of operation.

$P = cv^2f$.

if $f \uparrow \longrightarrow P \uparrow$.

Power loss: ~~due to~~

1. Switching power
2. Power loss during power ON state or OFF state. → depends on Device characteristics.

- ZVS (Zero Voltage Switching)
- ZCS (Zero Current Switching)

* Algorithmic level:

1. Programming quality.
2. Location of the memory. — (Memory Access)
3. Use of cache
4. Efficient algorithms. (w.r.t the HW)

* Architecture level:

→ Paralleling & Pipelining.

~~→ Gate & logic level~~

* Gate & logic level:

Cryptographic Algo → adder: CLA (static, CMOS, Domino, PTL), CLA. ⋮
Multiplier. ⋮

* Technology: 65 nm (Less power consumption)
180 nm (High " ")

* Control clk power Reduction:

FSM: Perfect.
Add components. (Combi + seq)

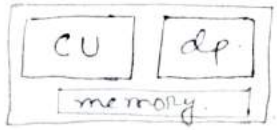
A K. Swain
12.03.14
Wed

[i] Single Purpose Processor & → ex: GCD.
[ii] General Purpose Processor.

[* For single Purpose Processor will be taught from F. Vahid book.]

[*] A processor always performs computational tasks. And it consists of —

i) datapath → [string, manipulating data ↓ ALU/shift...]
ii) Controller. ← [Moving data through the datapath]
iii) Memory RAM/ROM.



* Differences btw GPP & SPP:

[i] GPP does variety of computation where a SPP does only specific computation.

[ii] For a SPP no memory is required but GPP does have memory to store the variety of computation.

* Advantage of SPP:

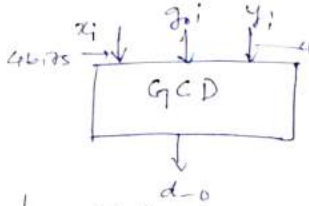
[i] Performance can be faster.
[ii] Smaller in size
[iii] Power consumption is less.
[iv] Time to market is less.

* Dis-adv:

[i] Cost is higher.

[11] Flexibility is less.

[+] Problem: Convert a computation task into a SPP.



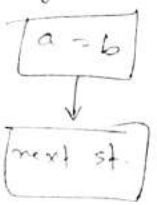
2. Convert the program (functionality) into a corresponding state diagram.

```
int x, y;
while (1) {
        while (!go_i);
          x = x_i;
          y = y_i;
        while (x != y) {
                if (x < y)
                    y = y - x;
                else x = x - y; }
        d_o = x;
}
```

eg: $x = 13$, $y = 5$
$x = 8$, $y = 5$
$x = 3$, $y = 5$
$x = 3$, $y = 2$.
$x = 1$, $y = 2$
$x = 1$, $y = 1$.

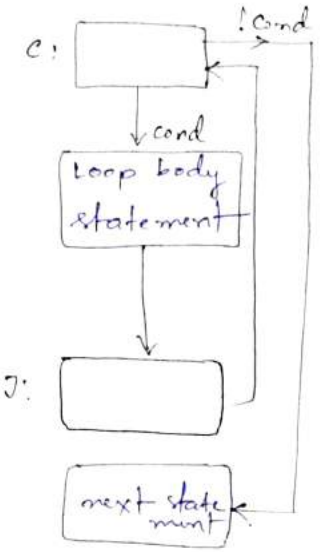
Terminology:

a) Assignment statement.

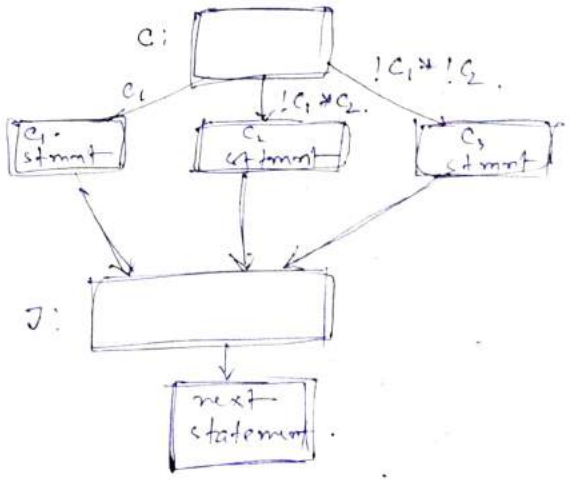


b) loop statement

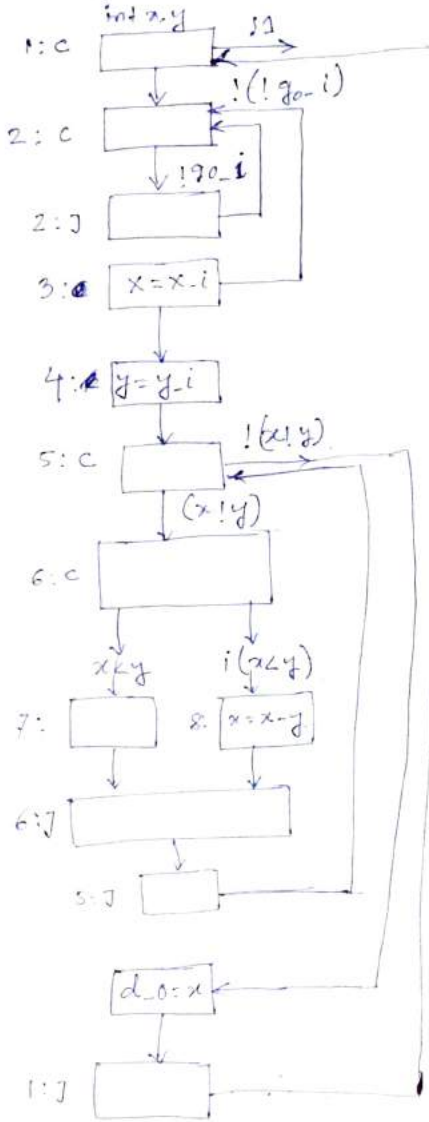
→ Next page



c) branch statement:



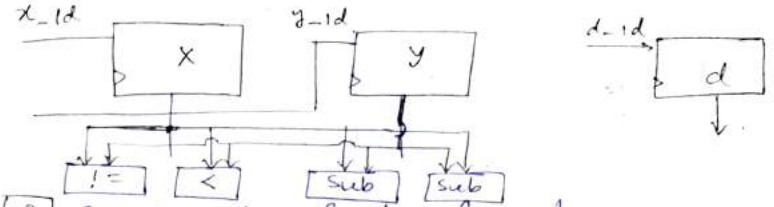
FSMD Diagram for the given program:



Thu
13-03-2014.

[4] Divide the functionality into datapath:

[1.] Create a register for any declared variable
i.e; x & y.
and also create register for all o/p
d_o;

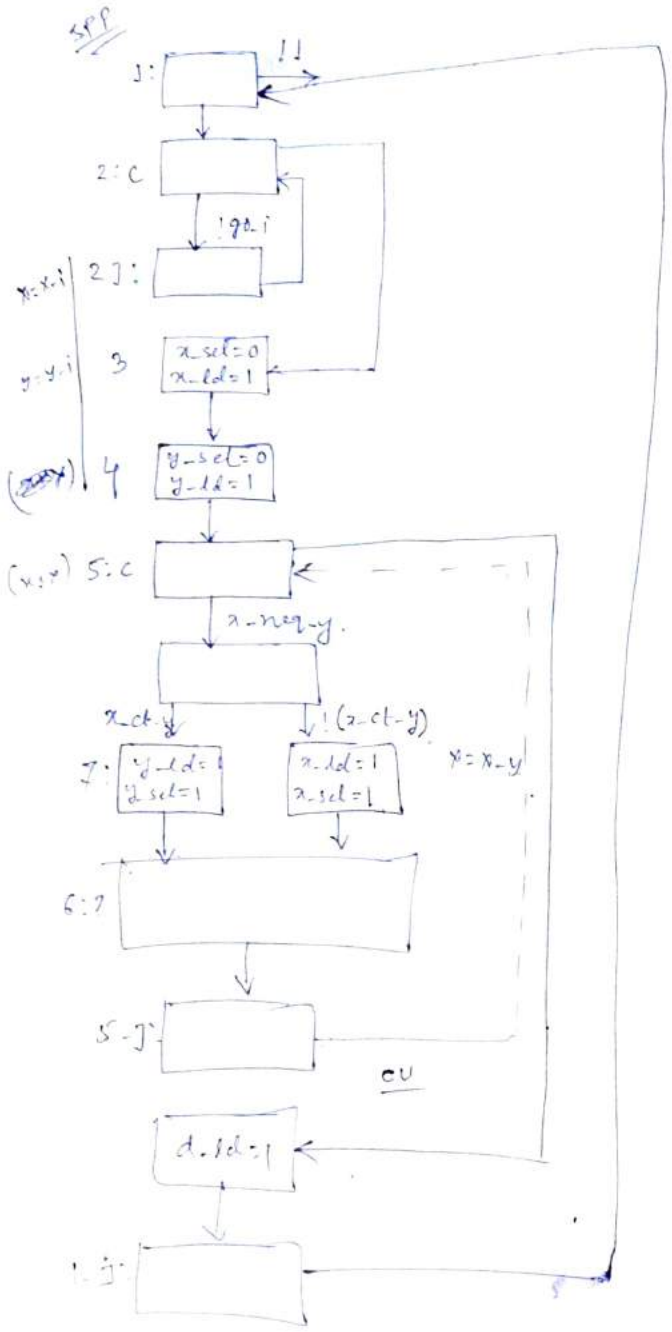


[2.] Create the functional unit:

2 - sub
1 - comparator
1 - comp Inequality.

[3] Connect points, registers, functional unit.

[4] Give an unique name (identifier) to all i/p, o/p's.



Fri
14-03-2014

[1] SPP: Optimization

1. Algorithm
2. FSMD
3. Datapath
4. State (FSM)

(Time complexity)

[1.] Algorithm: Analyzes the no of computation & size of variables, is being used in the algo. (Space complexity)

```
int x_y, R;
while (1)
    while (!go_i):
     if (x_i >= y_i)
        { x = x_i; y = y_i; }
      else
        {x = y_i; y = x_i;}
    while (y != 0)
      R = x % y;
      x = y.
      y = R;
    }
    d_o = x;
}
```

1.0; if x = 42 & y = 8 then,

* by using old method

x = 42 y = 8

①	x = x-y;	26	8
②	→	18	8
③	→	10	8
④	(x<y)	2	8
⑤		2	6
⑥		2	4
⑦		2	2
⑧	d_o =	2	

[8 clk pulses]

* by using

x: 42 y = 8

x = x%y = 2 x = y = 8, y = 2

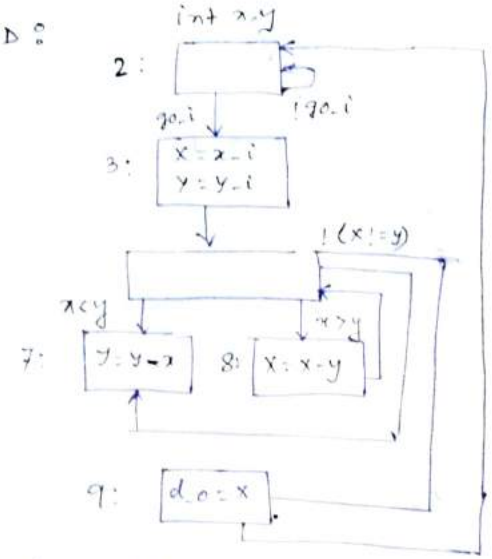
x = 8 y = 2

∴ x = x%y = 0 x = y = 2, y = 0

d_o = 2.

* speed ↑ size ↓

2. FSMD :



* Here 13 states reduces to ≠ 6 states

Scheduling: The task of assigning operations from original program to states of FSMD

* Assign: Do the soda machine problem.

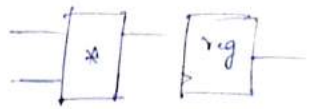
3. Datapath → One to One mapping.

RTL

2. a = b * c * d * e

1 2 3

[3 multipliers]



a = b*c*d*e.

Note: Instead of using 3 multipliers, we can use

(1)) 1 multiplier & 1 register (mux).

Speed will be less but the cost will be less as well bcoz we use 1 multiplier instead of 3. Pipelining can also be used.

(2.) We can use 1 subtractor with same arrangement of ~~1~~ register & MUX instead of 2 subtractors.

~~in place of~~

[cost ↓ speed ↓]

4. FSM: State Diagram.
State Table.
State encoding.
State minimization.

Wed
19.03.2014

GPP : - General Purpose Processor Design :

→ Does variety of computation.

8 bit → MPU
- → C.U
- → D.P
- → Memory & ROM.

Factors to decide GPP :-

[1] First Define its instruction set.

[2] How the instructions are encoded & executed.

[3] How many instructions do we want? What are these?

[4] What Opcode do we assign to each instruction

[5] How many bits do we use to encode an instruction?

[i] Instruction

[ii] Datapath

1. What functional units do we need?
2. How many registers do we need?
3. Do we use a single registers file. or seperate registers.

4. How the different units are connected together (MUX)

3. Control Unit : (CU)

→ It asserts the control signals to the datapath.

→ FSM → (i) Fetch an instruction
(ii) Decode the instruction.
(iii) Execute the instruction.
(iv) Appropriate control signals to do the operation.

8-bit MPU :

8 bit Datapath Unit :