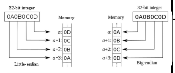
Miscellaneous
- Float:4,Long:4/8,Pointer:4/8,LL:8,Long Double:16
- 1word = 32Bit = 4Byte = 8HexDigit = 4Row
- LC2K: 32b Addr/Instr, ARM 64b Addr, 32b Instr, 32 Reg
- Little Endian: read, store Bottom-Up
- Offset for ARM branch instr. is #instr. (*4 to get Addr/line)
- Header of Linux ELF has size of text, data, ST & RT
- Output based on current state: Moore; cur. state&input: Mealy
- RISC has fixed instr.len, more memory, generally faster
- Moore:transis*2 two yrs. DennardScaling:energy remains same

Binary Operations



- X-bit Two's Complement $[-2^{X-1}, 2^{X-1})$
  1: 0001, 0: 0000, -1: 1111, -2: 1110
  $-A = \sim A + 1 = A$ nor $A + 1 = 0$ nor $A + 1$
- Bit setting k digits at *j-i*: val |= (0bXXXX<<i)
- Bit peeking k digits *j-i*: (val &= (0b111<<i)) >>= i, *k* of 1s

C-ARM

1. if (a % 2^n) → even
   ANDI X_{temp}, X_a, #$2^n - 1$
   CBZ X_{temp}, else / CMPI X_{temp}, #0
2. arr[i].var = arr[StructSAddr + i*size(arr) + varSAddrinStruct]
   LSL X_{temp}, X_{idx}, #$\log_2 sizeof(arr)$
   ADD X_{temp}, X_{temp}, #$X_{baseAddr}$
   LDUR X_{ele}, [X_{temp}, #varSAddrinStruct] //Alignment

In LC2K, *lw* R_i, R_{ele}, *BaseAddrofarr* [ONLY for arr of int]
For arr of struct of 2 int & access 2nd int, *lw* $2*R_i+1$, R_{ele}, *BA*

3. arr[i].val /= 2
   LSR X_{ele}, X_{ele}, #1
   STUR X_{ele}, [X_{temp}, #varSAddrinStruct]

Caller/Callee (Times #func execution)
- Caller: Locate all the assign. and set bkpt to split up the var liveness, if var not used in btw FC&assigning step, DON'T save. Caller save in leaf function. Ignore args, return. Loop var=#loop.
- Callee: 1 save at func start if appeared, ALWAYS 0 for main()

LC2K(anlsbjhn)

Type	Instr.	31-25 Unused	24-22 OPC	21-19	18-16	15-3	2-0
R	add	0	000	Reg A	Reg B	0	Reg Dst
	nor		001				
I	lw		010			OSF16bit [-32768, 32767]	
	sw		011				
	beq		100				
J	jalr		101			0	
O	halt		110	0			
	noop		111				

add: $R_{dst}.v=R_A.v+R_B.v$; nor: $R_{dst}.v= \sim(R_A.v \mid R_B.v)$; 0x00??000?
lw: $R_B.v$ = mem[$R_A.v$ + OSF].v; 0x00??????
sw: mem[$R_A.v$ + OSF].v = $R_B.v$; 0x00??????
beq: PC = ($R_A.v$ == $R_B.v$) ? PC + 1 + OSF: PC + 1; 0x01??????
jalr: $R_B.v$ = nextline.num; branch line.num = $R_A.v$; 0x01??0000
halt: ++PC and stop; 0x01800000, noop: ++PC; 0x01C00000
Label is only for OSF, for beq: label.v = label.linenum – (PC+1)

Linker

Symbol Table(No repeat)
- Any func(inc. main): only *T* when within file. NO func args.
- Var: extern(*U*), global/static/str literal(*D*).
  NO local, #def, #inc, struct declaration

Relocation Table(May repeat in diff. lines, ONLY look in func)
- Func Calls: Only *BL* when NOT within file
- Global Var(Ignore all local var): *LDUR* for RetVal,
  LDUR if it's used in file (inc. comparison),
  Z=X+Y: *LDUR* X&Y, *STUR* Z; *=, +=, ++: *LDUR*, *STUR*,
  strcpy(X,Y): *LDUR* X, *STUR* Y, *BL*
  NO *STUR* for static local initialization and *BL* for sizeof()

Floating Point $(1.4\times10^{-45}, 3.4\times10^{38})$
IEEE: +0/-1(1bit), exp(8bit), mantissa(23bit)
Decimal→IEEE: Conv. DEC to BIN, move dot to left by k digits. Exp = k+127, Mantissa = digits after dot (extend to 23bit)
IEEE→Decimal: Add implicit 1. to the front of mantissa.
Shift right dot by Exp-127 times and convert it to decimal

Struct Alignment (X-bit archi, affects pointers)
- Primitive: StartAddr is divi. by own size
- Struct: StartAddr & StructSize are divi. by largest atomic size
- Maximize by putting in size in ascending or descending order

MultiCycle LC2K Latency(FDEMW)

	RIM	RFR	ALU	RDM	WDM	WFR	Cyc
add nor	✓	✓	✓			✓	4
sw	✓	✓	✓		✓		4
lw	✓	✓	✓	✓		✓	5
beq	✓	✓	✓				4
jalr	✓	✓				✓	4
noop halt	✓						2

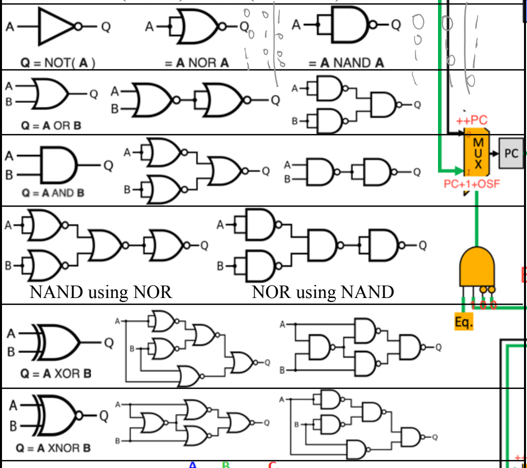
Bit Encoding
#STbits = $\log_2 \#states$
#ROMbits = $2^{\#INbits + STbits} \times$ (#OUTbits + STbits)

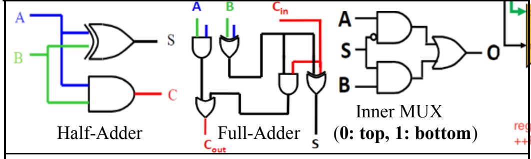
Execution Time/Runtime
= #Instr. × CPI × Clock Period = #Cycles × Clock Period/Cycle
= (#lw*5 + #noop/halt*2 + #rest*4) × Clock Period
average cycles/instrn

Logic & Gates
A^B = (A&~B) | (~A&B) = (A|B) & ~(A&B)
~(A^B) = (A|~B) & (~A|B) = (A&B) | ~(A|B)
NAND: 1110, NOR: 1000, XOR: 0110, XNOR: 1001

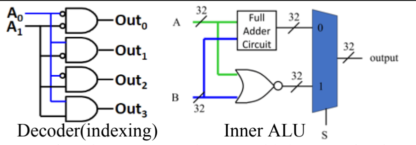
~a = a NOR a = a NOR 0 // a NOR 1 = 0
Let ***G=(a NOR b)***, ***H=(a NOR a)***, ***M=(b NOR b)***
a OR b = G NOR G // a AND b = H NOR M
a NAND b = (a AND b) NOR (a AND b)
a XOR b = (a AND b) NOR G
a XNOR b = (a NOR G) NOR (b NOR G)
= (b NOR H) NOR (a NOR M)

~a = a NAND a = a NAND 1 // a NAND 0 = 1
Let ***Q=(a NAND b)***, ***T=(a NAND a)***, ***W=(b NAND b)***
a AND b = Q NAND Q // a OR b = T NAND W
a NOR b = (a OR b) NAND (a OR b)
a XOR b = (a NAND Q) NAND (b NAND Q)
= (b NAND T) NAND (a NAND W)
a XNOR b = (a OR b) NAND (a NAND b)



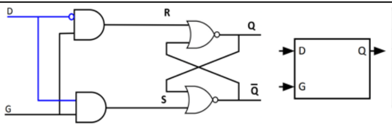


Half-Adder Full-Adder Inner MUX **(0: top, 1: bottom)**



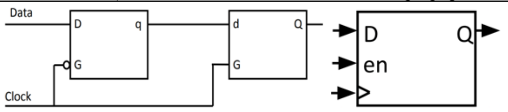
Decoder(indexing) Inner ALU

A:B decoder means getting an *A*-bit input and only activates the corresponding 1 output out of *B* possible outputs. In the case of ROM, it presets the value(that's used to control MUXes) with the corresponding opcode. Decoder then selects and activates the ROM line based on the input opcode.

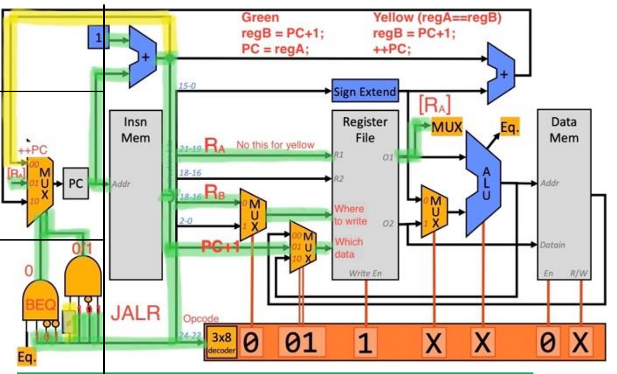


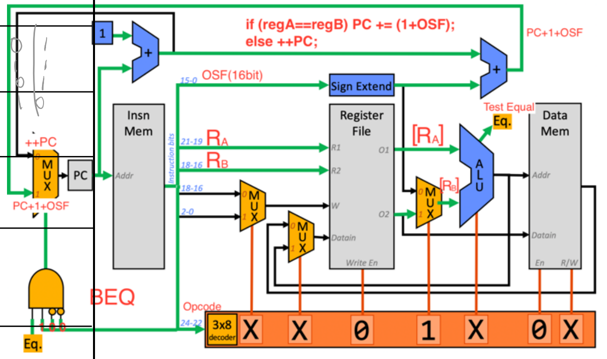
D	G	R	S	Q	!Q
0	0	0	0	Q	!Q
0	1	1	0	0	1
1	0	0	0	Q	!Q
1	1	0	1	1	0

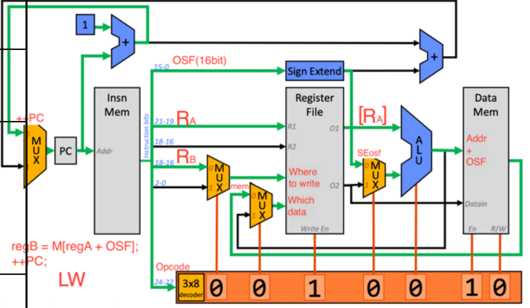
D Latch: Change in D is completely reflected in Q thruout the entire window when G is high. Copy exactly D for every corres. high G window & extend horizontally for low G(it rmbs last value it reads). It's unstable as fluctuation in D will propagate.

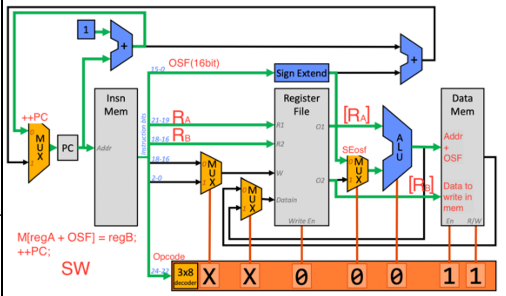


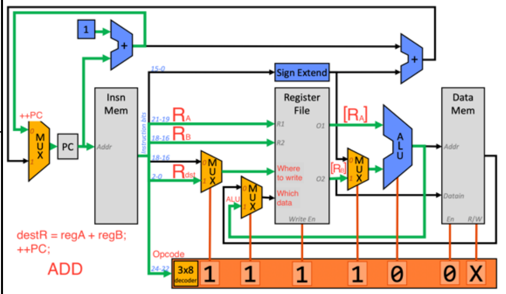
D FlipFlop: D is only read in at rising edge and keeps it constant till next rising edge (thruout one cycle). Extend the D level at start of clock all the way to end of clock.

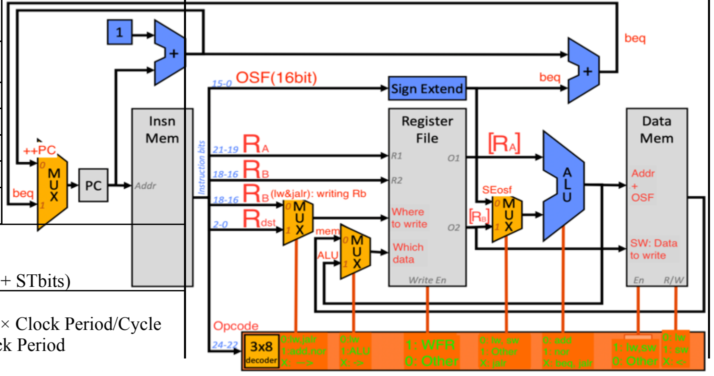












Control Signal of Single-Cycle ROM

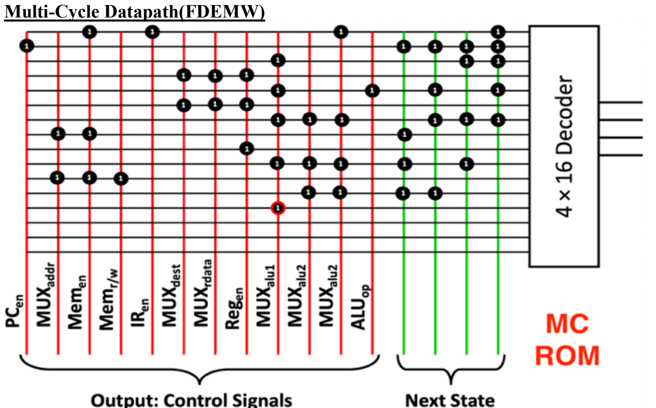
Meaning	0		1	
WFR to R_B/R_{dst}	R_B	lw, jalr	R_{dst}	add
WFR from Mem_{data}/ALU_{res}	Mem_{data}	lw	ALU_{res}	add
[from PC+1]	***0***		***1***	
WFR?	No	sw,beq	Yes	others
OSF/R_B	OSF	lw,sw	R_B	add,beq
ADD/NOR	ADD	lw,sw	NOR	
Touch Mem?	No	lw,sw	Yes	others
R/W data	Read	lw	Write	sw

int32_t i=0
data[i]
x0 data[]
x3 i
x5 = data[i]
LSL X4 X3 #4
ADD X5 X4 X0

n inputs
2^m states
k outputs
$2^{m+n} \cdot 2^k$

• $\% 2^9 = \cdot \& (2^9-1)$

Multi-Cycle Datapath(FDEMW)



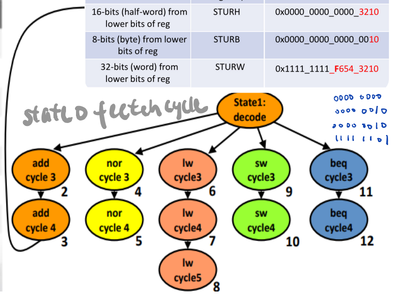
Unused (fill seven 0)							Opcode			Reg A			Reg B			Unused (fill thirteen 0)													Dst Reg		
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	?							0	0	0	0	0	0	0	0	0	0	0	0	0			
0				0												0				0				0							

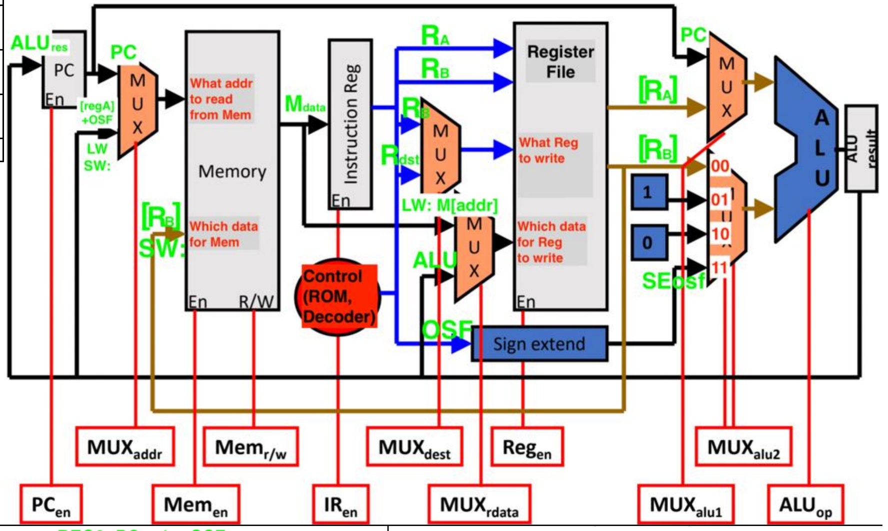
Unused (fill seven 0)							Opcode			Reg A			Reg B			Offset Field (2's complement)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0																									
0				0/1																											

Unused (fill seven 0)							Opcode			Reg A			Reg B			Unused (fill sixteen 0)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	1							0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0				1												0				0				0				0			

Desired amount of data to transfer?	Operation	Example
64-bits (double word or whole register)	STUR (Store unscaled register)	0xFEDC_BA98_7654_3210
16-bits (half-word) from lower bits of reg	STURH	0x0000_0000_0000_3210
8-bits (byte) from lower bits of reg	STURB	0x0000_0000_0000_0010
32-bits (word) from lower bits of reg	STURW	0x1111_1111_F654_3210

Stage	Name	Description
1	Base Case	If `(r==n) || (r==0)`, skip to stage 9 to return 1.
2	Callee-Stores	Store callee-saved registers (at least `r1`, `r2`, `r7`) to the `Stack`.
3	Function Call 1	Decrement `r1`, then `jalr` to the start of the function.
4	Caller-Store	Store the returned result to the `Stack`.
5	Function Call 2	Decrement `r2`, then `jalr` to the start of the function.
6	Caller-Load	Retrieve the result from Call 1 and add it to the result of Call 2.
7	Callee-Loads	Retrieve callee-saved registers from the `Stack`.
8	Return Sum	`jalr` back to the address stored in `r7`.
9	Return 1	Set `r3` to `1` and return.





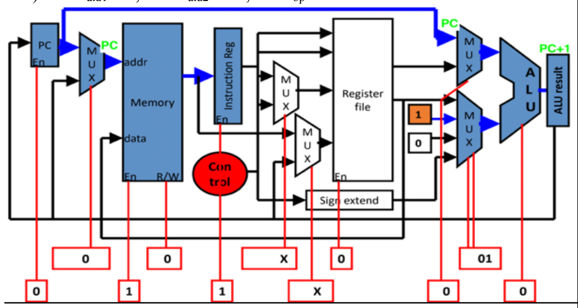
Output: Control Signals

Labels	Meaning	0		1	
PC_{en}	Write PC from ALU_{res}?	No		Yes: ONLY Decode	
MUX_{addr}	Read Mem[addr] from PC / R_A+OSF	PC		R_A+OSF: Only lw,sw C4	
Mem_{en}	Mem Instr?	No		Yes: Only Fetch, lw,sw C4	
$Mem_{r/w}$	Read/Write Mem	Read		Write	
IR_{en}	Write into IR?	No		Yes: ONLY Fetch	
MUX_{dest}	Write R_B/R_{dst}	R_B: Only lw C5		R_{dst}: Only add C4	
MUX_{rdata}	RFR from M[addr]/ALU_{res}	M[addr]: Only lw C5		ALU_{res}: Only add C4	
Reg_{en}	WFR?	No		Yes: Only add C4 & lw C5	
MUX_{alu1}	PC/R_A	PC		R_A: Only add,lw,sw C3, beq C4	
MUX_{alu2}	R_B / 1 / 0 / SEosf	R_B: (00)	1: (01)	0: (10)	SEosf: (11)
ALU_{op}	ADD/NOR	ADD		NOR	

1. Fetch: a) <u>RIM from PC and store into IR</u> b) **<u>Calculate</u>** <u>PC+1</u>

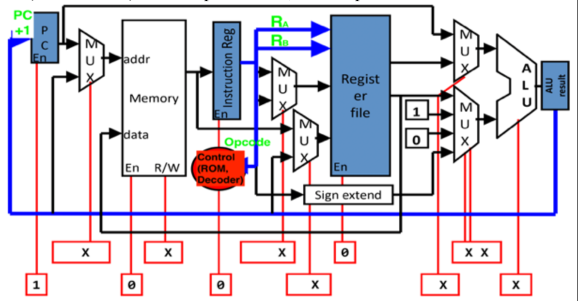
a) MUX_{addr} = 0, Mem_{en} = 1, $Mem_{r/w}$ = 0, IR_{en} = 1

b) MUX_{alu1} = 0, MUX_{alu2} = 01, ALU_{op} = 0

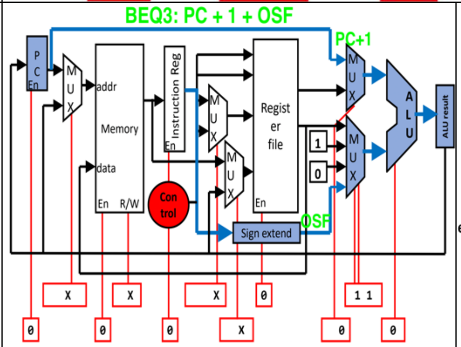


2. Decode: a) <u>Update/Write PC</u> b) <u>RFR from IR & change state with opcode</u>

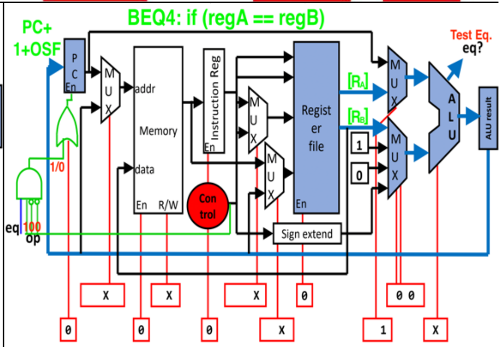
a) PC_{en} = 1 b) IR was split into R_A, R_B & Opcode



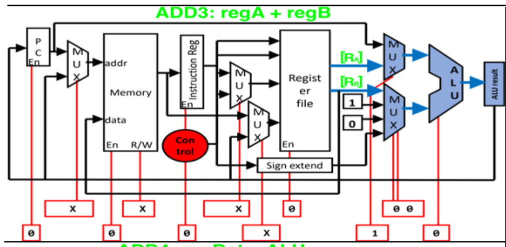
BEQ3: PC + 1 + OSF



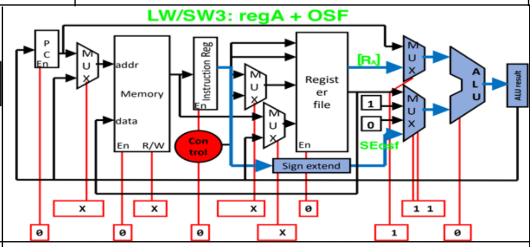
BEQ4: if (regA == regB)



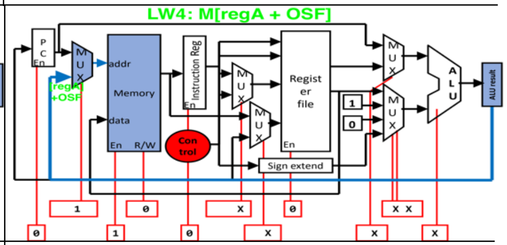
ADD3: regA + regB



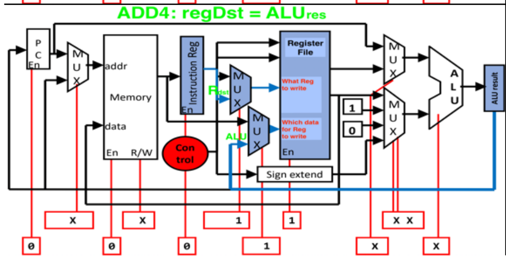
LW/SW3: regA + OSF



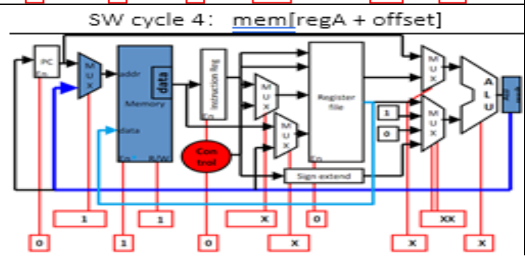
LW4: M[regA + OSF]



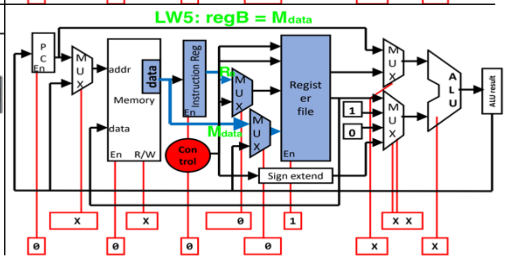
ADD4: regDst = ALUres



SW cycle 4 : <u>mem[regA + offset]</u>



LW5: regB = Mdata



bits byte word

LDUR X3, [X4, #100]

- Load (unscaled) to register—retrieve a double word (64 bits) from address (X4+100)

LDURH X3, [X4, #100]

- Load halfword (16 bits) from address (X4+100) to the low 16 bits of X3—top 48 bits of X3 are set zero

LDURB X3, [X4, #100]

- Load byte (8 bits) from address (X4+100) and put in the low 8 bits of X3—zero extend the destination register X3 (top 56 bits)

What about loading words?

LDURSW X3, [X4, #100]

- retrieve a word (32 bits) from address (X4+100) and put in lower half of X3—top 32 bits of X3 are sign extended