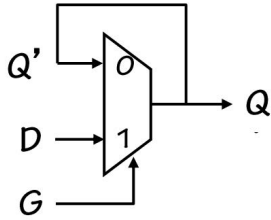


Computational Structure - Cheatsheet - W3-W5

Week 3

D-Latch

Mux with a feedback loop



represents the clock signal
 -When G is 1, D is selected in the mux. (Write Mode)
 -When G is 0, Q follows Q'. (Read memory mode)

Possible Problems

Storing invalid information: If G changes from 1 to 0 at the exact moment when D is changing (invalid), the new invalid information of wire D will be stored instead
 Unstable Output because of unstable input: In practice, input from users can be highly unstable.

The Dynamic Discipline (addresses problem 1)

The input to a synchronous sequential circuit must be stable during the aperture (setup and hold) time around the clock edge. The dynamic discipline states that

1. $T_{setup} = 2t_{pd}$
2. $T_{hold} = t_{pd}$

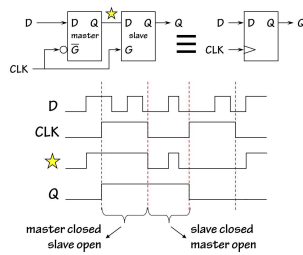
1. T_{setup} = the minimum amount of time that the voltage on wire D needs to be stable before the clock edge changes from 0 to 1.
2. T_{hold} = the minimum amount of time that the voltage on wire D needs to be stable after the clock edge changes from 0 to 1.
3. t_{pd} is the propagation delay of the D-latch

Flip-Flop

Created by putting 2 D-latches in series

1. There is an inverter on the G input on the master flip flop
2. When CLK signal is 0, the G wire of master latch receive a 1 while slave flip flop receive a 0
 Master latch : Write mode
 Slave latch : Read memory mode
3. Vice-versa when CLK signal is 1
4. 1 D-Latch is on write mode why 1 D-latch is on memory mode at anytime.

Flip Flop Waveforms



Note that output wire Q only changes when CLK rises from 0 to 1

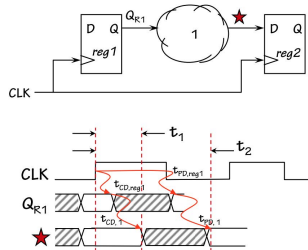
Timing Constraint

1. t_{CD} of a D-latch is the time taken for invalid CLK input to produce an invalid output on wire G
2. T_{PD} of a D-latch is the time taken for valid CLK input to produce a valid output on wire G

Flip-Flop Timing Constraint

- $t_{CD_{master}} > t_{HOLD_{slave}}$
1. When signal at G in master changes from 1 to 0, and signal at G in slave changes from 0 to 1, master goes into read memory mode and slave goes into write mode.
 2. Again, these CLK signals do not magically transform from 0 to 1 or 1 to 0. It has to gradually change to high voltage '1' or to low voltage '0', and it will cross the invalid region (the region which voltage value doesn't translate to either digital 1 or 0).
 3. Then Q_{star} cannot change too quickly while G is transitioning, otherwise it will not meet the hold time of the slave latch.
 4. This means the contamination delay of the master latch has to meet the hold time of the slave latch.

Sequential Logic Timing Constraint



$$t_1 = t_{CD,reg1} + t_{CD,1} > t_{HOLD,reg2}$$

$$t_2 = t_{PD,reg1} + t_{PD,1} < t_{CLK} - t_{SETUP,reg2}$$

$$t_{work} = t_{PD}$$

Metastable State

Properties

1. It corresponds to an invalid logic level
2. Unstable equilibrium which will settle to valid 0 or 1 eventually
3. Settling can be arbitrarily long
4. All bistable system exhibits at least 1 metastable state
5. Cannot be avoided but can be minimize

Clock Skew

$$t_1 = t_{CD,R1} + t_{CD,R1} >$$

$$t_{HOLD,R2} + t_{skew}$$

$$t_2 = t_{PD,R1} + t_{PD,CL1} <$$

$$t_{CLK} - t_{SETUP,R2} + t_{skew}$$

Week 4

Programmable Machines

Programmable control system

1. Control processing at each step with FSM
2. Allow different control sequences to be loaded into control FSM
3. Re-use data path and reconfigure FSM to compute new function

Short-comings

1. Tiny repertoire of operation
2. Unable to generate and execute a new program
3. Limited storage

Finite State Machine:

Enumeration

FSM with i inputs, o outputs, s states

1. Truth table has 2^{i+s} rows with $(o + s)$ columns each
2. $2^{(o+s)} 2^{i+s}$ max state
3. Limitation : cannot solve problems with arbitrarily many states

Turing Machines

Turing Machine Specification

1. Doubly-infinite tape
2. Discrete symbol positions
3. Finite alphabet
4. Control FSM
 Inputs - Current Symbol
 Outputs - Write 0/1, move Left/Right
5. Initial Starting State S_0
6. Halt StateHalt

Properties

1. Can be used to compute integer functions of form $y = T_k[x]$
2. Where k : FSM index, x : input tape configuration, y : output tape configuration.
 *Not all integer functions can be computed with Turing Machines
3. Computable functions : $f(x)$ computable
 $\Leftrightarrow \exists k : \forall x : f(x) = T_k[x] = f_k(x)$
4. Church-Turing Hypothesis states that any computable function is computable by a TM

Universal Functions and Universality

Universal function: $U(k, j) = T_k(j)$

U is comtable by a Turing Machine

→ k encodes a 'program'

→ j encodes the input data to be used

→ T_u interprets program

Von Neumann Model

4 components

1. CPU - contains several registers as well as logic
2. Memory = storage of N words with W bits, where W is a fixed architerctural paramter, and N can be expanded to meet needs
3. Input/Output
4. Connection Bus

Week 5

Machine Language and Compilers

Compiler

1. Compiler translate high-level language into low-level assembler machine language
2. Done before execution and slows program development
3. Decisions made during compile time,before execution

Interpreter

1. Computes exact instructinos
2. Done after execution and slows down program execution
3. Decisions made during run time, after execution

UASM

1. LONG tells the assembler to assemble a 32 bit quantity

Stacks and Procedures

1. Add one item at a time to the top of the stack by PUSH, and remove one item at a time from the top of the stack by POP
2. Stack pointer (SP) points to the available memory location to write to (first unused stack space). SP is actually the content of R29.
3. base pointer (BP) points to the base of the stack, or equivalently first item pushed to the stack.
4. There are only 2 operations to modify a stack: Push and Pop
5. In the 'original state' of the diagram, the SP is set to address 0x0000 1000
6. Push is a 2 step procedure:
 - Increase value of SP by 4 byte
 - Writing data to SP - 4
7. Pop is also a 2 step procedure:
 - store data at addressed pointed by SP
 - reduce value of SP by 4 byte

Some Beta Documentation

- BEQ/BF - If Ra = zero, the PC is loaded with the target address;
 literal = ((OFFSET(label) - OFFSET(current instruction)) / 4) - 1
 $PC \leftarrow PC + 4$
 $EA \leftarrow PC + 4 * SEXT(literal)$
 $TEMP \leftarrow Reg[Ra]$
 $Reg[Rc] \leftarrow PC$
 if $TEMP = 0$ then $PC \leftarrow EA$
- JMP -
 $PC \leftarrow PC + 4$
 $EA \leftarrow Reg[Ra]$ and 0xFFFFFCC
 $Reg[Rc] \leftarrow PC$
 $PC \leftarrow EA$
- LDR - The effective address EA is computed by multiplying the sign-extended literal by 4 (to convert it to a byte offset) and adding it to the updated PC. The location in memory specified by EA is read into register Rc. The Ra field is ignored and should be 11111 (R31). The supervisor bit (bit 31 of the PC) is ignored (i.e., treated as zero) when computing EA.
 literal = ((OFFSET(label) - OFFSET(current instruction)) / 4) - 1
 $PC \leftarrow PC + 4$
 $EA \leftarrow PC + 4 * SEXT(literal)$
 $Reg[Rc] \leftarrow Mem[EA]$
- Stack pointer (SP) points to the available memory location to write to (first unused stack space). SP is actually the content of R29.

Finding Setup Time for Input

1. CASE 1: INPUT $-i$ REGISTER
 $t_S = t_{SR1}$
2. CASE 2: INPUT $-i$ CL $-i$ REGISTER
 $t_S = t_{pdCL1} + t_{SR1}$

Finding Hold Time for Input

1. CASE 1: INPUT $-i$ REGISTER
 $t_H = t_{HR1}$
2. CASE 2: INPUT $-i$ CL $-i$ REGISTER
 $t_H = t_{HR1} + t_{cdCL1}$

Propagation and contamination delay of circuit

Propagation delay : The time taken to produce a valid output after the CLK rise turns valid

Contamination delay : The time taken to produce an invalid output after the CLK turns invalid

1. CASE 1: REGISTER $-i$ OUTPUT
 $t_{cd} = t_{cdR1}$
2. CASE 2: REGISTER $-i$ CL $-i$ OUTPUT
 $t_{pd} = t_{pdR1} + t_{pdCL1}$
 $t_{cd} = t_{cdR1} + t_{cdCL1}$

Finding the minimum CLK period

According to timing constraint t2, the clock period has to be larger than the time taken to finish the 'work' (propagation delays plus the setup time of the downstream register) between two registers. (Diagram on back-end)








- 1. At each clock period (each time the clock rise from 0 to 1), a new input is being "loaded" to the registers, and the previous input is passed through to the rest of the components downstream
- 2. So before the next clock rise, one has to finish all the necessary work downstream until the next register, in this case the next register is R2.
- 3. There are two paths, blue and red where the output of R1 will flow downstreams.
- 4. The time taken for the blue path to 'work' (propagation delays plus setup time for the first register downstream blue path) is:
 $t_{blue} = t_{pd_{R1}} + t_{pd_{CL1}} + t_{pd_{CL3}} + t_{S_{R3}}$
- 5. The time taken for the red path to 'work' (propagation delays plus setup time for the first register downstream red path) is:
 $t_{red} = t_{pd_{R1}} + t_{pd_{CL2}} + t_{S_{R2}}$
- 6. Both blue path and red path 'work' has to be done before the next clock rise
- 7. Hence minimum CLK period is max of t_{blue}, t_{red}

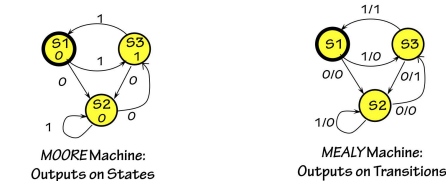
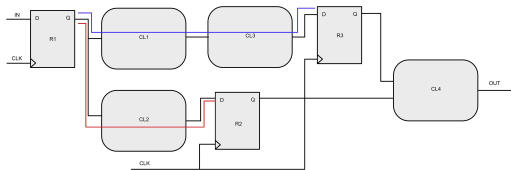
Hex	Binary	Octal	Decimal
0		0	0
1		1	1
2		10	2
3		11	3
4		100	4
5		101	5
6		110	6
7		111	7
8		1000	8
9		1001	9
A		1010	12
B		1011	13
C		1100	14
D		1101	15
E		1110	16
F		1111	17
10		1 0000	20
11		1 0001	21
24		10 0100	44
5E		101 1110	136
100		1 0000 0000	400
3E8		11 1110 1000	1750
1000		1 0000 0000 0000	10000
FACE		1111 1010 1100 1110	175316
			64206

Writing Assembly Language Procedure

- 1. Calling Sequence - Arguments - (Write) push arguments into the stack in the reverse order. write a constant into the register using CMOVE, and then push it into the memory.
- 2. Calling Sequence - Branching and Cleanup - (Write) - call the function itself using BR , and then write cleanup codes (Deallocate).
- 3. Standard Entry Sequence
PUSH(LP)
PUSH(BP)
MOVE(SP,BP)
- 4. The Actual Code
- 5. Exit Sequence - Pop Regs from Actual Code
MOVE(BP,SP)
POP(BP)
POP(LP)
JMP(LP)

Logic Gates

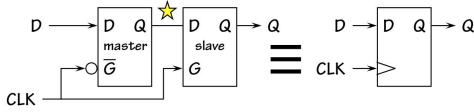
Name	NOT	AND	NAND	OR	NOR	XOR	XNOR																																																																																																
Alg. Expr.	\bar{A}	AB	\overline{AB}	$A+B$	$\overline{A+B}$	$A\oplus B$	$\overline{A\oplus B}$																																																																																																
Symbol																																																																																																							
Truth Table	<table><tr><th>A</th><th>X</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	X	0	1	1	0	<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	X	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	X	0	0	1	0	1	1	1	0	1	1	1	0	<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	0	<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	X	0	0	1	0	1	0	1	0	0	1	1	1	<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	0	<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	X	0	0	1	0	1	0	1	0	0	1	1	1
A	X																																																																																																						
0	1																																																																																																						
1	0																																																																																																						
A	B	X																																																																																																					
0	0	0																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					
A	B	X																																																																																																					
0	0	1																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
A	B	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
A	B	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					
A	B	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
A	B	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					



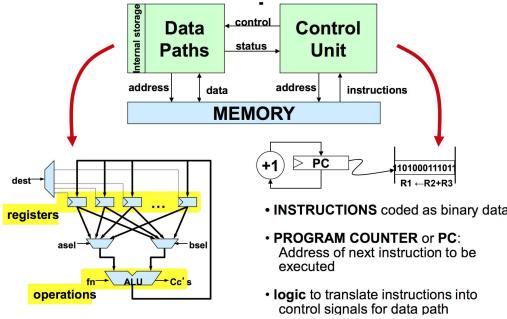
- Arcs leaving a state must be:
- (1) mutually exclusive
 - can't have two choices for a given input value
- (2) collectively exhaustive
 - every state must specify what happens for each possible input combination. "Nothing happens" means arc back to itself.



IN	Current State	Next State	Unlock
0	SX 000	S0 000	0
1	SX 000	SX 000	0
0	S0 001	S0 001	0
1	S0 001	S01 010	0
0	S01 011	S0 000	0
1	S01 011	S011 010	0
0	S011 010	S0110 000	0
1	S011 010	SX 000	0
0	S0110 000	S0 001	0
1	S0110 000	S01 011	0



OPCODE 6 bits	Rc 5 bits	Ra 5 bits	Rb 5 bits	11 bits unused
Type 1				
OPCODE 6 bits	Rc 5 bits	Ra 5 bits	16 bits signed constant	
Type 2				



Macro	Definition
BEQ(Ra, label)	BEQ(Ra, label, R31)
BF(Ra, label)	BF(Ra, label, R31)
BNE(Ra, label)	BNE(Ra, label, R31)
BT(Ra, label)	BT(Ra, label, R31)
BR(label, Rc)	BEQ(R31, label, Rc)
BR(label)	BR(label, R31)
JMP(Ra)	JMP(Ra, R31)
LD(label, Rc)	LD(R31, label, Rc)
ST(Rc, label)	ST(Rc, label, R31)
MOVE(Ra, Rc)	ADD(Ra, R31, Rc)
CMOVE(c, Rc)	ADDC(R31, c, Rc)
PUSH(Ra)	ADDC(SP, 4, SP), then ST(Ra, -4, SP)
POP(Rc)	LD(SP, -4, Rc), then SUBC(SP, 4, SP)
ALLOCATE(k)	ADDC(SP, 4*k, SP)
DEALLOCATE(k)	SUBC(SP, 4*k, SP)

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

Example: Factorial

Beta assembler code:

fact:	PUSH(LP)	save linkages
	PUSH(BP)	
	MOVE(SP,BP)	new frame base
	PUSH(r1)	preserve regs
	LD(BP,-12,r1)	r1 ← n
	BNE(r1,big)	if (n != 0)
	ADDC(r31,1,r0)	else return 1;
	BR(rtn)	
big:	SUBC(r1,1,r1)	r1 ← (n-1)
	PUSH(r1)	push arg1
	BR(fact,LP)	fact(n-1)
	DEALLOCATE(1)	pop arg1
	LD(BP,-12,r1)	r1 ← n
	MUL(r1,r0,r0)	r0 ← n*fact(n-1)
rtn:	POP(r1)	restore regs
	MOVE(BP,SP)	Why?
	POP(BP)	restore links
	POP(LP)	
	JMP(LP,R31)	return.

