University of Puerto Rico – Mayagüez
School of Engineering

INEL 4206 – Microprocessors

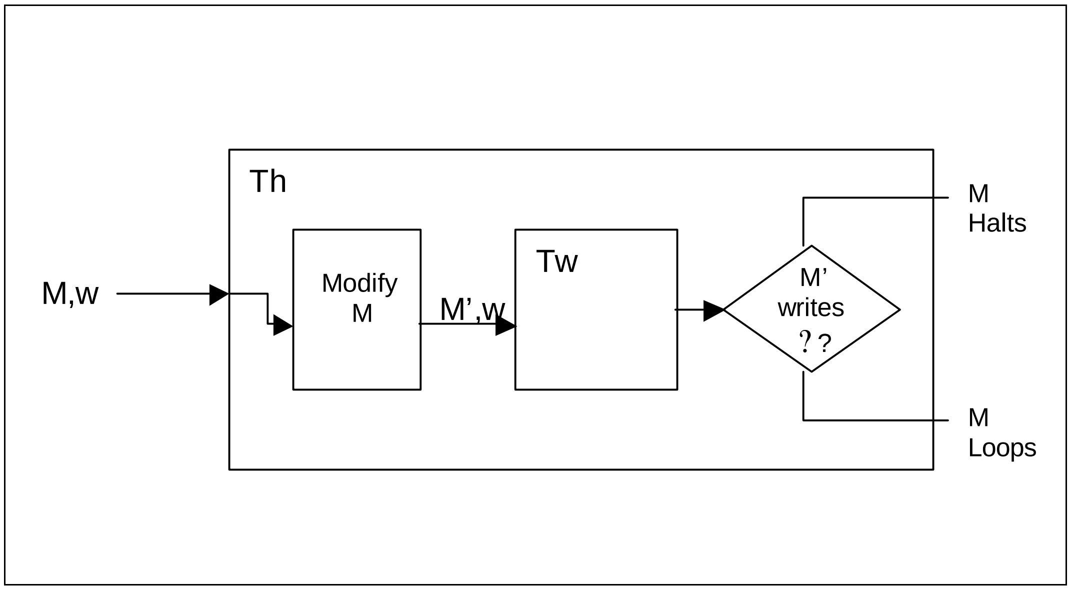
Problem Set 1 – Due February 15, 2002

1. Consider the problem of determining if a Turing Machine ever writes a specific symbol from its alphabet into its tape.

    a. Argue that this problem is undecidable.

We can argue by contradiction that a TM, say Tw, capable of solving this problem cannot exist. The argument first assumes that Tw exists. Then we argue that if this assumption is true, then we can compute the Halting Problem. Since we already know that the Halting Problem is undecidable, then it must be true that our only assumption is false.

To argue that Tw can be used to solve the Halting problem we can design a new Turing Machine Th as shown in the following diagram. Th Takes a TM M and an input tape w and determines whether or not M halts on input w. To accomplish this, Th modifies the description of M adding a previously unused symbol ? to its alphabet and a new state. The modified version of M, call it M', transitions into the new state and writes the new symbol ? whenever M would enter its halt state. It should be clear that M' will write ? to tape if and only if M would have entered its halting state. Therefore, we Th can solve the Halting problem.



b. Try to generalize this result to other problems concerning the algorithmic determination of properties of Turing Machines.

Many interesting properties about programs are impossible to detect algorithmically.

2. Consider an ALU with two n-bit inputs (A and B) and with the following operation table:

Operation	Selection code	Ouput
PASS A	0 0 0	A
ADD	0 0 1	A+B+ Ci
NOP	0 1 0	Don't Care
SUB	0 1 1	A – B - Ci
AND	1 0 0	A and B
XOR	1 0 1	A xor B
OR	1 1 0	A or B
COMPARE	1 1 1	Not (A xor B)

a. Using the bit-slice design technique provide a schematic diagram of a 1 bit ALU segment including its internal gate-level logic.

PDF file with diagram of 1 Bit ALU available on web site.

b. The circuit made in part a, can be connected to another of the same type to form a n-bit ALU. To do this we have to separate the input into the N bits of A, the N bits of B and give the results in N bits plus a Carry Out. To complete the model, we connect the Carry Out (C_o) of each 1-Bit-ALU to the Carry In (C_i) of the next, the C_i of the first ALU (calculating the least significant bit) should be connected to a 0V or otherwise and external Carry Signal, and the C_o of the last (most significant ALU) will be returned to the circuit using the ALU. Such implementation and tests of operation are attached in the following pages.

PDF file with diagram of N-Bit ALU available on web site.

3. Consider the design of the Easy I Architecture discussed in class:

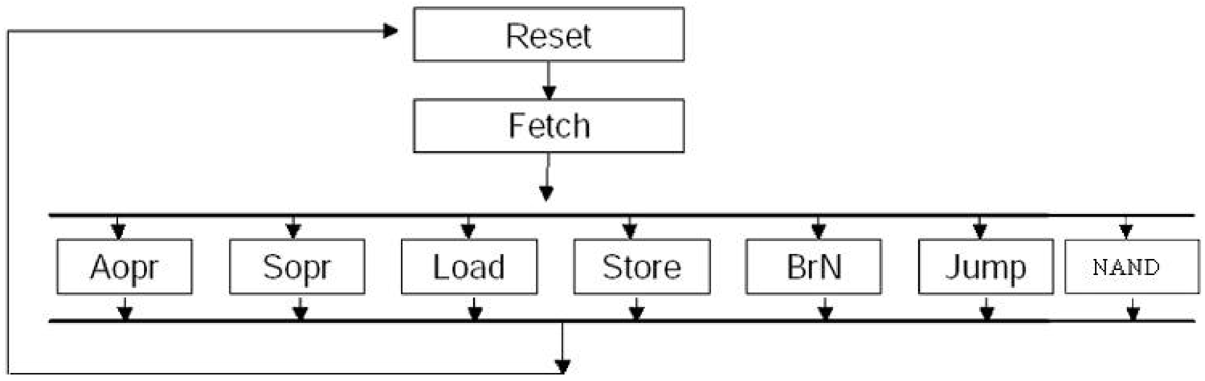
a. Add a NAND instruction that computes the bitwise logical NAND operation as described by the following RTL expression:

AC <- AC NAND MEM[X]

Include diagrams illustrating the necessary changes to the Easy I flowcharts and control unit design.

Data Path stays the same

Control Unit Flowchart



Level 2 Flowchart for NAND

NAND 1

DI<0..9> ✍ ABUS->A0	

?

NAND 2

A0 ✍ EAB	▒
EDB ✍ DI	

?

NAND 3

DI✍ ABUS✍ ALU	
AC✍ ALU_B	
ALU✍ AC	

?

NAND 4

AC✍ ALU_B	
ALU✍ AC	
PC✍ A0	
PC✍ PC+2	

?

FETCH

Control Unit Stat Transition Table Addition

Current State	opcode	AC:15		Next State	ALU Op	Mem Op	PC sel	PC is	DI LE	AC le	A0 sel	A0 le	EDB sel
Fetch	xx xxx	x		NAND1	xxx	RD	11	x	1	0	x	0	x
NAND1	01 000	x		NAND2	xxx	NOP	11	x	0	0	1	1	x
NAND2	xx xxx	x		NAND3	xxx	RD	11	x	1	0	x	0	x
NAND3	xx xxx	x		NAND4	AND	NOP	11	x	0	1	x	0	x
NAND4	xx xxx	x		Fetch	NOT	NOP	10	1	0	1	0	1	x

Current Encoding for states consists of 4 – bits, to make this new instruction we must add an additional bit, so that there are enough states available.

NEW 5-bit Encoding for STATES

State	Encoding
reset1	00000
reset2	00001
fetch	00010
aopr	00011
sopr	00100
store1	00101
store2	00110
store3	00111
load1	01000
load2	01001
load3	01010
brn1	01011
brn2	01100
jump	01101
NAND1	01110
NAND2	01111
NAND3	10000
NAND4	10001

b. There are several tradeoffs for providing this special instruction directly on the processor. First we must add the instruction NAND by giving it as a new choice to FETCH, which will add one more option at the time of deciding where to go. Second we must provide a circuit path (shown here as a flowchart) for the steps to take in order to compute the NAND. One of the most important changes that have to be made is the addition of a bit for the state encoding, since with the available 4 is not enough to include the 4 new states.

When we include this new instruction we can have a faster access to the result of a NAND call than we would have if we were to use AND and NOT separatly, since we don't have to load the AC twice, it does the computation without returning to Fetch and going through the decision process again. But we are faced with adding new components to the procesor (ex. the additional bit) which can make it slower.

To decide if it is an intelligent choice to include this instruction we would have to know how frequently it would be used. If it is commonly used then it would be wise to have it included in the hard-coded instructions, but if it is scarcely used then it would not be wise to inlcude it on the processor's instruction set.