CE1921 HOMEWORK HW9	SPRING 2019	NAME: Julian Singkham

- This homework is **due on printed paper at the start of the <u>last</u> lecture in Week 10.**
- Your solution for each problem must show your work to receive partial credit. Failure to show your thought process will not earn points for incorrect answers.

1. Which signal **<u>is not</u>** sampled by IDEX in the **pipelined** CE1921 ARM processor?

   A. REGWR B. ALUSRCB C. REGSRC D. [✓ EXTS] E. MEMWR

   Which signal **<u>is</u>** sampled by MEMWB in the **pipelined** CE1921 ARM processor?

   A. [✓ REGWR] B. REGDST C. ALUS D. MEMWR E. ALUSRCB

   Which register value **<u>is</u>** sampled by EXMEM in the **pipelined** CE1921 ARM processor?

   A. RD1 B. PC C. R14 D. [✓ RD2] E. PC4

   Which signal **<u>is not</u>** sampled by EXMEM in the **pipelined** CE1921 ARM processor?

   A. [✓ REGDST] B. REGWR C. MEMWR D. REGSRC E. ALUF

2. Programmers often want to invert the bits of a register. High-level languages include operators to specify bit inversion. For example, the C programming language allows a programmer to write:

```
j = ~k;     // invert all bits of variable k
```

The limited instruction set given on the course cream-colored reference card would allow implementation of this operation as EOR of k with the value 0xFFFFFFFF. This would require a sequence of instructions discovered by students in the last homework assignment. One solution i:

```
mov R0,#0   ; begin forming -1 = 0xFFFFFFFF
sub R0,R0,#1; 0 - 1 = -1 = 0xFFFFFFFF
eor R0,R0,#1; complete the inversion: R0 xor 0xFFFFFFFF = not R0
```

Multiple step solutions consume multiple clock periods and reduce overall performance. The ARM architects took the approach of "making the common cases fast." Bitwise inversion is **very common** in low-level systems programming of device drivers, hardware control routines, etc. Thus, the ARM ISA has had a single-cycle solution to bitwise complement since the first version (ARMv1). This instruction is **move-the-not (MVN)**. Its **formal** ARM syntax is:

```
MVN<cond>{S} Rd,Rm{,<shift>}
MVN<cond>{S} Rd,#imm
```

With ALUS select F
    <= not INTB when "101"

I picked 101 because ALUS 101 is INTA which is not used

allowing a flexible set of instructions such as:

```
MVN   R0,#0 ; form 0xFFFFFFFF = -1
MVNS  R0,#A ; form 0xFFFFFFF5 = -6, sample CVNZ
MVN   R1,R2 ; copy the inverse of R2 to R1
MVNS  R1,R2 ; copy the inverse of R2 to R1, sample CVNZ
MVNNE R9,R4 ; copy the inverse of R4 to R9 if Z=0
MVNEQ R5,R2 ; copy the inverse of R2 into R5 if Z=1
```

These instructions improve performance on a very common task by reducing the **clocks-per-instruction** from the multiple clock cycles required previously to the one clock cycle required for the MVN instruction.

Consider the instructions MVN and MVNS only. **Describe** the changes you would need to make to the controller and the ALU of the CE1921 **basic** single-cycle processor to add these instructions.

3. **Evaluate** this code segment and **identify** all hazards as Rn or Rm in the provided hazard table. **Remember** that in CE1921, all registers sample on the sample edge that the program counter uses to advance. Thus, write-back data will not store into the register file until the rising-edge of the clock at the end of the write-back period. This means that the instruction must complete write-back before the stalled decoding instruction can access it in decode.

```
MOV   R1,#42
SUB   R0,R1,#5
LDR   R3,[R0,#18]
STR   R4,[R1,#63]
ORR   R2,R0,R3
```

	1	2	3	4	5
1	▨	▨	▨	▨	▨
2	Rn	▨	▨	▨	▨
3		Rn	▨	▨	▨
4	Rn			▨	▨
5		Rn	Rm		▨

4. **Calculate** the total execution time, in units of time (µs, ms, s, etc.) for this code segment on the CE1921 ARM single-cycle processor if the clock frequency is 1MHz. **Show all work** to receive partial credit.

```
      MOV   R0,   #5
      MOV   R1,   #0
L1    CMP   R0,   R1
      BEQ   DONE
      SUB   R0,   R0,   #1
      B     L1
DONE  MOV   R12,R0
```

2

4 instructinos * 5 iterations=20 + 2 instructions (cmp + BEQ leads to DONE)

1

2+20+2+1=25 instructions / 1Mhz= 25µs