# CS 251, Spring 2023, Assignment 3

Due Thursday, June 29, 10:00 PM
Late submission accepted until Friday, June 30, 10:00 PM with no penalty.

**You are required to read, complete and sign, and submit (as well as follow) the following statement of Academic Integrity.**

Statement of Academic Integrity for CS 251 Spring 2023, **Assignment 3**
I declare the following statements to be true:

- I have not used any unauthorized aids.
- I recognize that while I can discuss the questions in this assignment on Piazza and other forums with the instructors and with other students in the class, the write up that I am submitting is my own.
- I am aware that misconduct related to course work can result in significant penalties, including failing the course and suspension (this is covered in Policy 71:)

  `https://uwaterloo.ca/secretariat/policies-procedures-guidelines/policy-71`

**Student Name:**

**UW ID#:**

**Signature:**

**Date:**

The purpose of this assignment is to help you understand the implementation of the single cycle processor.

Coverage material for this assignment can be found on the course website as:

- On the Lecture Notes webpage in the
  - 
  - June 20th Lecture on ROM, Register Files, SRAM and DRAM,
  - June 22nd Lecture on Single Cycle Processor Datapath
  - ARM reference

For this assignment, make note of the following details:

- Any diagrams that are part of your solution must be drawn neatly, using rectilinear lines and clearly labelled inputs and outputs.

We have an A3 Official Post on piazza. In this post, we will include all A3 assignment related information such as

- updates/corrections made to the assignment, if any
- remark request due date
- FAQs

1. (5 points) Consider the single-cycle computer shown on page 12 of this assignment for this question.

   Suppose the circuit elements take the following times:

| **ALU Component and Operation** | **Time** |
|---|---|
| Read Instruction Memory | 100ps |
| Register Read | 30ps |
| Register Write | 30ps |
| ALU and all adders | 50ps |
| Read or Write Data Memory | 100ps |

   You may assume that all other datapath components not listed in the table above take negligible time, e.g. PC, Control Unit, ALU Control, and all MUXes.

   Compute the **minimum** cycle time for each instruction type below:

   R-format: _____ps.

   LDUR: _____ps.

   STUR: _____ps.

   CBNZ or CBZ: _____ps.

   Branch: _____ps.

2. (12 points) Consider the ARM assembly language instruction at memory address 5016:

```
5016: SUB X10, X30, X11
```

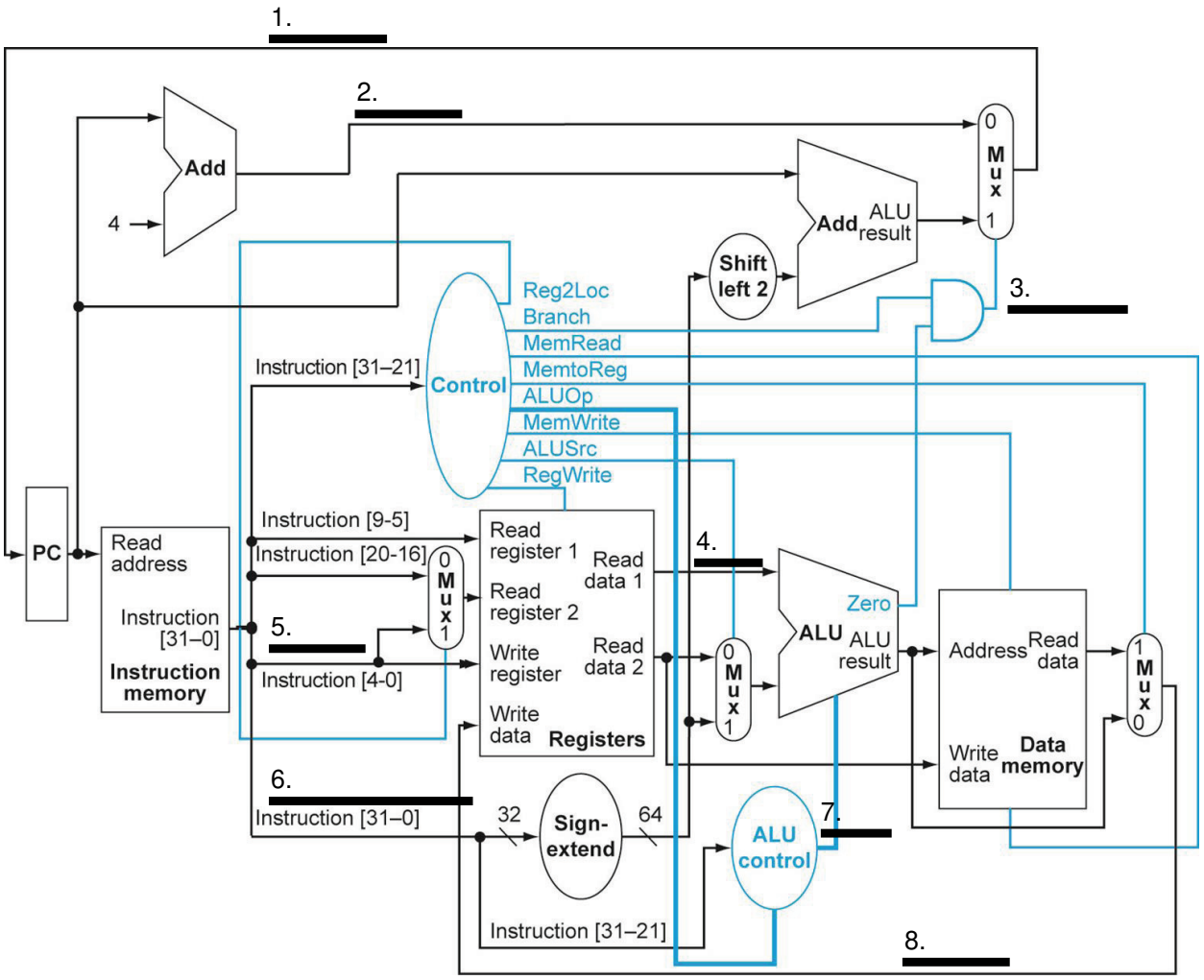
(a) (8 points) In the figure on the next page, there are eight labels. On each label, write in the value that travels along the corresponding wire(s) when executing this assembly language instruction.

Note: you should write a decimal number on each dark line, and **not** an expression involving things like 'PC', etc. Some numbers are more natural to write in binary; for any binary numbers you use, you should subscript them with a "2" like $101_2$. Assume that each register $X_i$ (for $0 \leq i \leq 30$) contains the decimal value $200_{10}+i$. Further assume that the `shamt` field of this R-format instruction is 0.

(b) (4 points) An ARM instruction is a 4-byte word. Complete the table below to represent the values of each of the 32 bits that encode this instruction. You will find it helpful to refer to the R-format depicted in the reference sheet on page 13.

| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | |

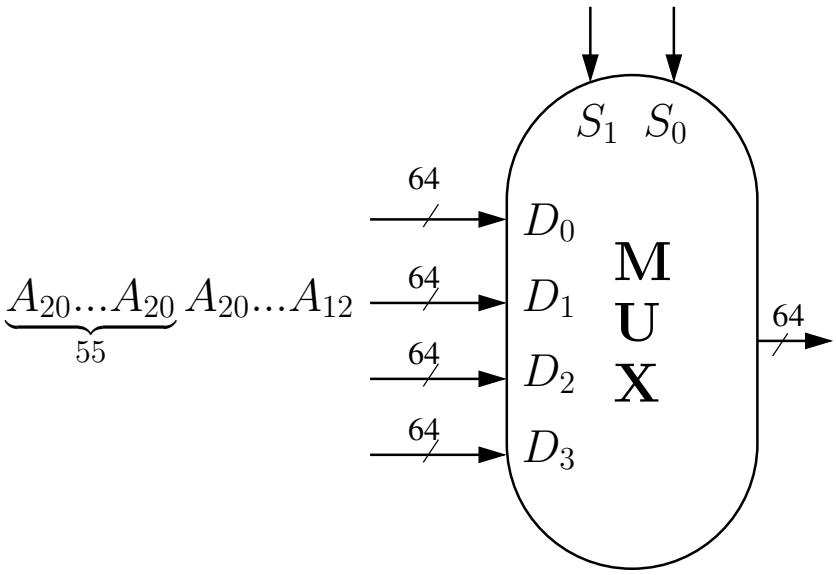

1.
2.
Add
4
0
M
u
x
1
Add
ALU result
Shift left 2
3.
Reg2Loc
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–21]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [9-5]
Instruction [20-16]
Instruction [4-0]
5.
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
4.
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
6.
Instruction [31–0]
32
Sign-extend
64
ALU control
7.
Instruction [31–21]
8.

3. (5 points) I-format, D-format, B-format, and CB-format instructions all have constant fields that get extended to 64-bits. The I-format constant field is treated as an unsigned number, while the other three formats treat their constant field as a 2's complement, signed number.

   Implement the sign-extension hardware found in the single cycle datapath. Your sign-extension hardware should take two inputs: the 32-bit instruction $A$, whose $n$th bit is referenced as $A_n$, and a 2-bit $SE$ input that indicates the format of the instruction:

| $SE_1 SE_0$ | Format |
|---|---|
| 0 0 | I-format |
| 0 1 | D-format |
| 1 0 | B-format |
| 1 1 | CB-format |

   Implement the sign-extension hardware using the 64-bit wide, 4-to-1 MUX shown below. We have completed one of the inputs for you. Complete the remaining inputs, and be sure to label the input to each select line.



4. (15 points) Consider the following possible ARM assembly language instructions:

```
STB  X1, [X2,#100]:   STB  Rd, [Rn,#100]  #Store and Branch
ORRI X1, X2, #100:    ORRI Rd, Rn | #100   #Bitwise OR with Immediate
CBU  X2, X3, #10:     CBU Rd, Rm, #10     #Conditional branch and update
```

The `STB` (store and branch) is an I-format instruction, `ORRI` is an I-format instruction, and `CBU` (conditional branch update) is an I2-format instruction (discussed below). The effect of each instruction is given below.

These new instructions **do not** require any physical hardware changes to the datapath.

Fill in the table below indicating the value of all existing control lines necessary to execute the `STB`, `ORRI` and `CBU` instructions on the datapath.

You **must use don't cares** where appropriate.

**The `STB` instruction** writes to memory and updates PC based on offset. Its effects are summarized as follows:

```
M[Rn + Offset] <= Rd
PC = PC + Offset x 4
```

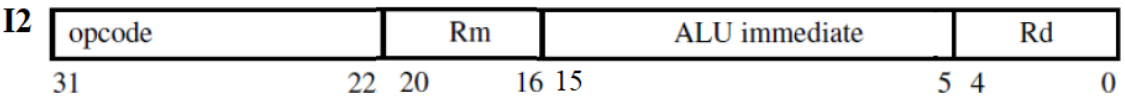
**The `ORRI` instruction** computes the bitwise OR of the contents of register Rn and the immediate constant. The result is stored in register Rd.Its effects are summarized as follows:

```
ORRI: Rd <= Rn | Immediate
PC = PC+4
```

**The `CBU` instruction** takes the value in register Rm and stores this into Register Rd. Its effects are summarized as follows:

```
Rd = Rm
if Rm == 0
then PC = PC + Offset x 4
```

You may assume for this question that `CBU` is an I2-format instruction that specifies register Rm in bits 20–16, that specifies the constant in bits 15–5, and that the Sign-Extend Unit has been extended to handle the constant in the I2-format. I2-format is given below:



| Type | Reg2 Loc | ALU Src | Mem ToReg | Reg Write | Mem Read | Mem Write | Brch | Uncnd Brch | ALU op1 | ALU op0 |
|---|---|---|---|---|---|---|---|---|---|---|
| R-format | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| `LDUR` | X | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| `STUR` | 1 | 1 | X | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| `CBZ` | 1 | 0 | X | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| `B` | X | X | X | 0 | 0 | 0 | X | 1 | X | X |
| `STB` | | | | | | | | | | |
| `ORRI` | | | | | | | | | | |
| `CBU` | | | | | | | | | | |

5. (10 points) We want to modify the single-cycle computer to implement the R-format instruction `ADDB`, or "add and branch". This ARM instruction has the following format:

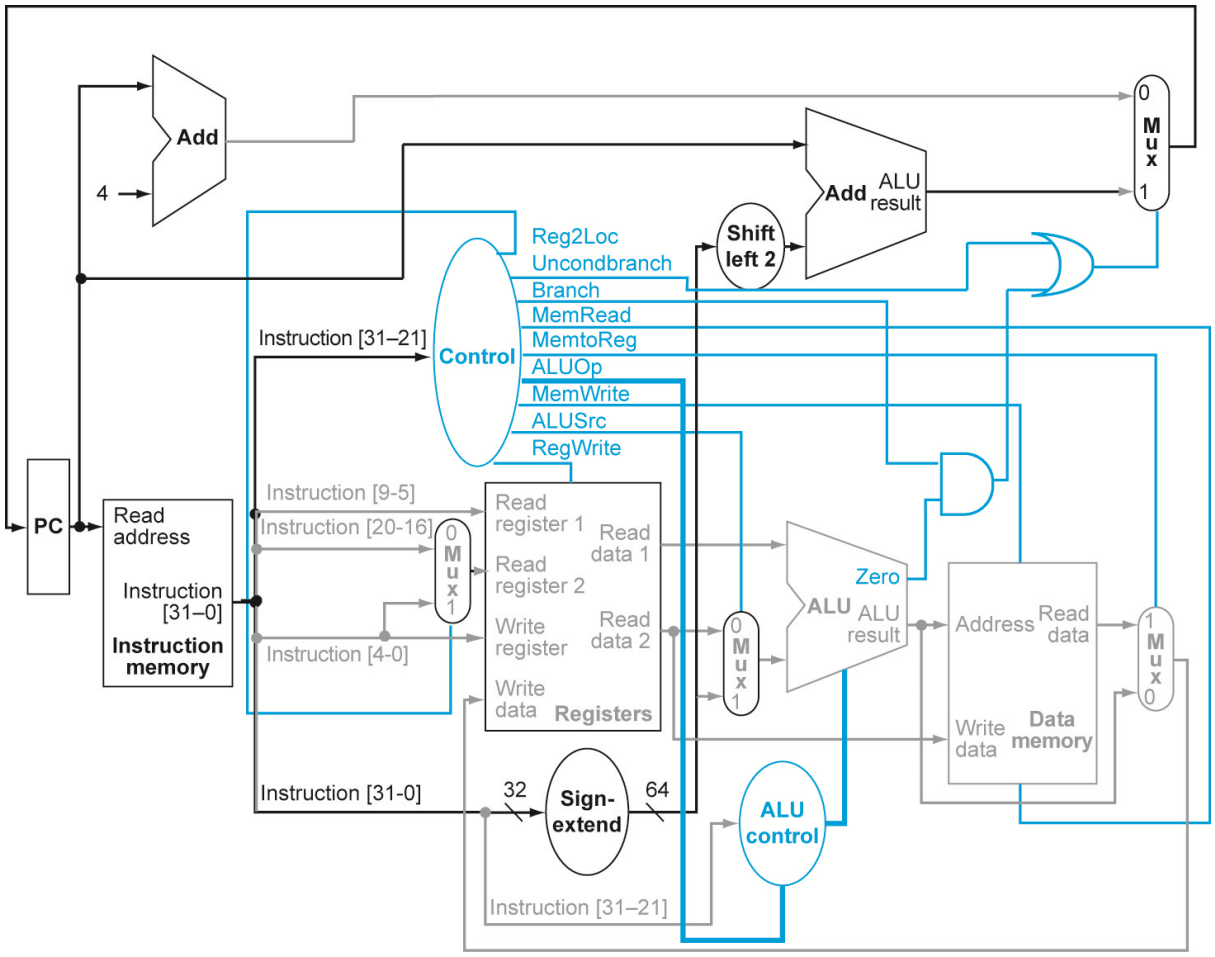
```
ADDB rd, rn, rm
```

This instruction adds the contents of registers `rn` and `rm`, stores the result in register `rd`, and branches so that $PC = PC + rn + rm$.

(a) (6 points) Modify the Single Cycle datapath on the next page to implement the R-format instruction `ADDB` to work as described above. You may add multiplexors, control bits and additional components as needed. You may assume the `ADDB` instruction will generate a unique control bit **BL**.

**Be sure that all other ARM instructions executing on the datapath still work.**

Summarize your changes to the datapath below, and make the modifications to the datapath provided on the next page.


Add
4
Add
ALU result
Shift left 2
M u x
0
1
Reg2Loc
Uncondbranch
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–21]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [9-5]
Instruction [20-16]
Instruction [4-0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
Instruction [31-0]
32
Sign-extend
64
ALU control
Instruction [31–21]

(b) (4 points) In the table below, give the settings of the control bits to implement the new `ADDB` ARM instruction. Use Don't Cares where appropriate. If you need an extra control line to implement this instruction or if you need to increase the number of bits in a control line, add additional columns to the table for the new control line, split a column to increase the number of bits in a control line, and in either case include a note below explaining the effect of the new/increased control line(s) on the datapath and what its setting should be for other instructions.

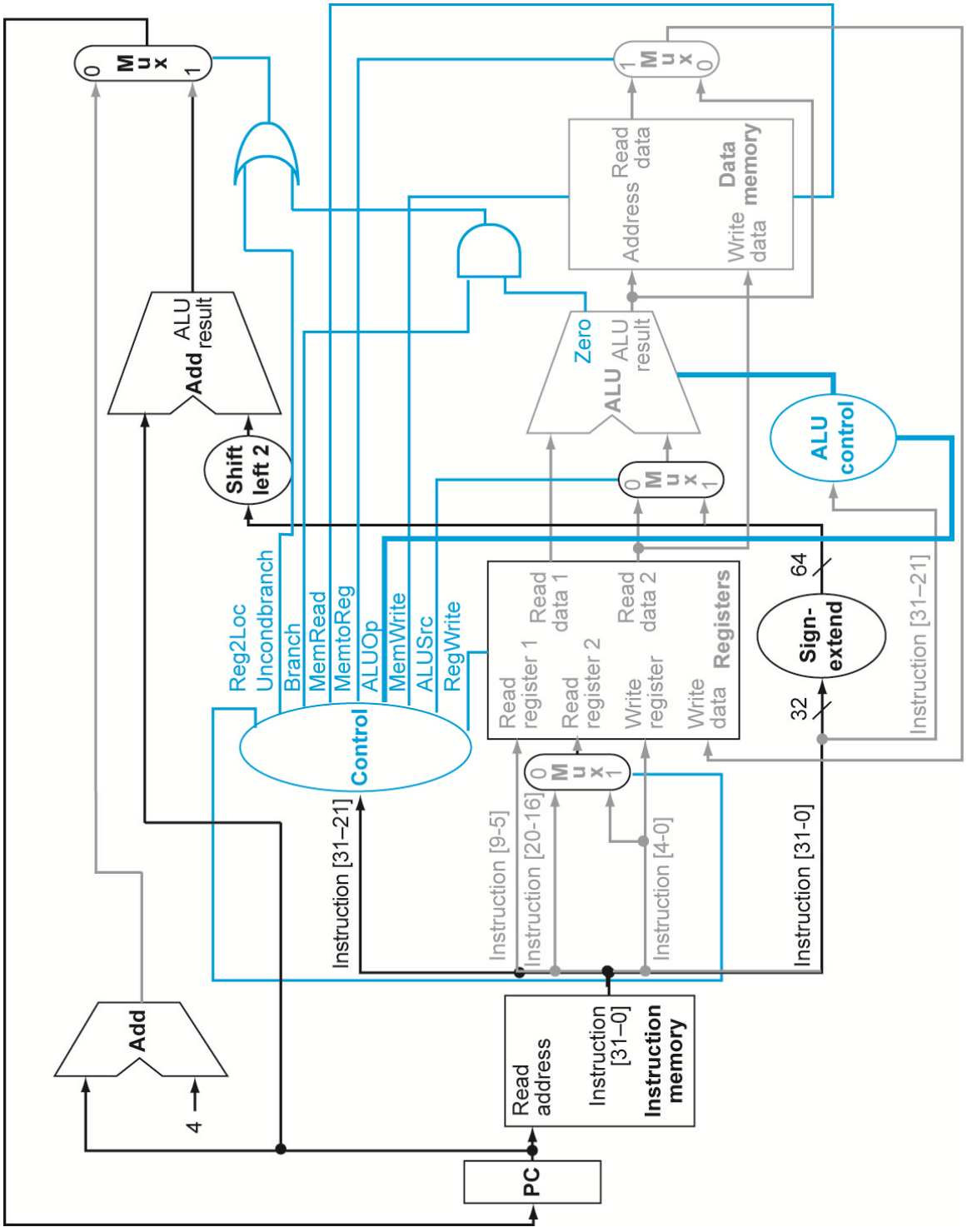
Make sure you do not break any other instructions. You should be able to determine the purpose and effects of each of the control signals from the Single Cycle datapath on page 12.

The following main control outputs have been abbreviated for convenience, `Branch` is Brch and `Uncondbranch` is Uncnd Brch

| Type | Reg2 Loc | ALU Src | Mem ToReg | Reg Write | Mem Read | Mem Write | Brch | Uncnd Brch | ALU op1 | ALU op0 | BL |
|---|---|---|---|---|---|---|---|---|---|---|---|
| `ADDB` | | | | | | | | | | | |

State the value(s) of any new control signal(s) for other ARM instructions:

A Single-cycle Processor



## ARM Instruction Type and Format