LSU EE 4720 **Homework 2** Solution **Due: 29 February 2008**

For the answers to these questions look at the ARM Architecture Reference Manual linked to the course references page, `http://www.ece.lsu.edu/ee4720/reference.html`.

Problem 1: The register fields in ARM instructions are four bits and so only 16 integer registers are accessible. The ISA manual describes ARM as having 32 integer registers, however many of them are only accessible in particular modes.

An advantage of fewer registers is that extra bits are available in the instruction encoding, for example, ARM three-register instruction formats would have three more bits available than the MIPS type R format. Where in the ARM formats do you think these bits went? In your answer give the instruction field and its purpose. There should be no equivalent in MIPS.

Every instruction format uses a `cond` (condition field), there is no counterpart to this in MIPS. The condition field is used to *predicate* instructions, that is, control whether or not an instruction has any effect.

(See section A3-1, which conveniently lists the instruction coding for many instructions.)

Problem 2: In MIPS an arbitrary 32-bit constant can be loaded into a register using a `lui` followed by an `ori`. In ARM the immediate field for data-processing (integer) instructions is only 8 bits.

(*a*) Show ARM code to put an arbitrary 32-bit constant into a register without using a load instruction. Use as few instructions as possible. *Hint: take advantage of ARMS shift and rotate capabilities.*

A move followed by three or's with shifts can do the trick.

```
 # Solution:
 # Note: The arbitrary constant is 0x12345678
 mov r1, # 0x78, 0
 orr r1, r1, # 0x56, 12
 orr r1, r1, # 0x34, 8
 orr r1, r1, # 0x12, 4
```

(*b*) Show how ARM can put an arbitrary constant into a register with one load instruction, whereas in MIPS two would be required. The MIPS code is shown below. Do not assume the address of the constant is **already** in a register, that would make this problem insultingly easy! *Hint: Use one of ARM's special purpose registers.*

```
.text
 lui r1, 0x1111
 lw r1, 0x2220(r1)
 # ... a few more instructions ..
 jr $ra
 nop

.data
my_32_bit_constant:  # Address: 0x11112220
.word 0x12345678
```

Solution shown below. As in the MIPS example the constant is stored in memory near the code. MIPS code requires two instructions, one to load the high 16 bits of the address, the second to load the data (using the load offset for the low 16 bits of the address). In ARM the program counter is one of the data processing (general purpose in MIPS) registers, `r15`. This makes something like a `lui` unnecessary in ARM because the program counter can serve as the load's base

register. The code below is in pseudo assembly language, the assembler would convert `-8 + my_32_bit_constant - HERE` into the correct offset.

```
# Solution
HERE:
 ldr r1, [r15 - 8 + my_32_bit_constant - HERE ]
```

Problem 3: In ARM the program counter is register `r15`, and so as far as instruction encoding goes, is treated as a general-purpose register.

(*a*) Why would really keeping the program counter in the integer register file add to the cost of an implementation?

Because to maintain an execution rate of one instruction per cycle one would need an additional read port and an additional write port on the register file to accommodate the program counter.

(*b*) How does the ISA manual hint that blue parts of the implementation below is what they had in mind? (Register `r15` is not stored in the register file, it will always be bypassed from the real PC.) (Note: The ARM implementation is far from complete and parts may not work.)

Because instructions that use their source operands in the EX stage (ordinary arithmetic and logical instructions and address operands for loads and stores) get not the value of PC, but PC+8, which is what you'd get in the diagram below. According to the ARM ISA stores of PC might result in PC+12 being written, which is consistent with the memory stage being three stages ahead of IF.