2.5 Program Counter related Addressing Modes

Note to students: *Some important concepts to bring across are:*
1. *Understand how PC relative addressing can support position independent code.*
2. *Understand how the modification of PC contents affects program flow.*
3. *Understand how to compute the offset relative to the PC to access the data memory area*

Figure 4 shows an ARM assembly program and the starting address of various incomplete instructions in code memory. Complete the mnemonics **M1** to **M5** based on their respective comments and ensure that all your solutions support **position-independent code**. You may use the partially completed VisUAL ARM assembly program template ***Tutorial 2_5-Template*** to test out your answers.

Note: The **PC** points 8 addresses ahead during the execution of each instruction as this is a consequence of how the ARM processes instructions using "pipeline" techniques.

Suggested solution:

Address	Mnemonics or Hexa Data	Comments
0x000	MOV R0,#0	; Clear R0
0x004	ADD PC,PC,#0	; **M1** - Relative Jump to instruction at addr 0x00C
0x008	MOV R1,R0	; Dummy instruction
0x00C	ADD R1,PC,#0xEC	; **M2** - Get start addr Var_100 into R1 in a PC-relative manner
0x010	LDR R0,[R1,#4]	; **M3** - Move the content at memory variable Var_104 into R0.
0x014	ADD R2,PC,#-4	; **M4** - Get start address of next instruction into R2.
0x018	MOV PC,R2	; **M5** - Create an infinite loop with this instruction.
:	:	
0x100	0x01234567	; Var_100 - Constant stored in data mem at address 0x100
0x104	0x89ABCDEF	; Var_104 - Constant stored in data mem at address 0x104

Figure 4_Ans – Various ARM mnemonics and their respective start addresses in code memory

Comments on solutions:

M1 – When the instruction at address 0x004 executes, the PC will be pointing 8 location ahead. This means it is already pointing to the instruction at 0x00C. In order to make a relative jump to that instruction, we just need to ADD zero to the current PC value. Adding to the current PC value is the way to make a relative jump.

M2 – In order to make the code position independent, we must assume that we have no knowledge of where the actual address the data we want to access resides. We can only access the data with the knowledge of its offset from the current PC position. As such, we must employ PC relative addressing. Position independent program (i.e. consisting of both executable code and its associated memory data) can be shifted anywhere in memory and still execute correctly.
The offset is computed by subtracting the address where the data is stored (i.e. 0x100) with the current PC value during the execution of the ADD instruction at 0x00C. Since the PC is always 8 location ahead during execution, the PC value during execution of the ADD instruction is (0x00C + 0x008) = 0x014. The PC relative offset to the memory variable at 0x100 is therefore given by 0x100 - 0x014 = **0x0EC**.

M3 – Since the start address of the memory data area is already in register R1 as a result of the instruction at M2, we need to add a positive offset of 4 to the base address in R1 in order to get to the next 32-bit memory variable Var_104. We can do this using register indirect with base plus offset.

M4 – In order to get address of the next instruction, we need to remember that the PC will be 8 addresses ahead during the execution of the instruction. So making a copy of the PC will not get us the next instruction but the address of two instructions away. In order to get the address of the immediate next instruction, we will need to subtract 4 from the current PC value. The ADD instruction is used to add -4 to the current PC value before moving the total summed value into destination register R2.

M5 – In order to get an instruction to execute in an infinite loop, it has to jump back to itself. Since we have instruction M4 to load the start address of the next instruction into R2, we can make the next instruction jump back to itself by making it load the value in R2 into the PC. Both instruction M4 and M5 are position independent since no knowledge of the actual address where these instructions were stored is actually employed.