Bryan Guner

* There are 13 fully general purpose registers inARMv7
* R13=Stack Pointer= register that stores address of last instruction on the stack. Decremented on push, incremented on pop.
* R14=Link Register=holds address to return to when a function call completes.
* R15=Program Counter=contains address of current instruction to be executed. (PC=PC+4Bytes)
* Size of coded machine instructions in ARMv7 ISA is :can be 16 or 32 bit
* # of address bits required to represent a ?-byte address space= log2(size of address space in bytes)
* Size of address space of n-bit address =2n : a 30-bit address would generate a GB, 31->2GB
* Big Endian: word address=address of MSByte | Little Endian =address of LSByte|ARM is little E
* Bit sizes & unsigned representable range: 1bit [0:1] | 4bit=1nibble [0:15] |8bit=1byte [0:255] | 16bit=1 half-word [0:65535]| 32bit=1word[0:232 -1] |64bit=1double-word [0: 264 -1]
* Bitwise Boolean Operators: AND=1 when both bits are 1| OR=1 if one of the bits are 1| XOR=1 if the bits are inverse of each other| NAND=1 when both bits have value of 1.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Bin | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 |
| Hex | 9 | A | B | C | D | E | F | --------- | -------- |
| Bin | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 | --------- | -------- |

* The s in ADDS stands for set flags, ADDS can be used for both signed& unsigned
* First 4 parameters in an ARMv7 procedure go to registers r0,r1,r2,r3, any extra go on stack, return value goes in r0
* To do binary subtraction using the borrow method: when you sub 0-1 borrow a 10 from the 1 in a higher digit (10-1=1 bc 10 is bin representation of 2); to sub a bigger # from a smaller # simply swap the #s, do the subtraction, and invert the result.
* Example(convert procedure to ‘C’ code): copy the null-terminated byte string starting at address ‘src’ to a memory array starting at address ‘dst’ ,make sure to copy terminating ‘0/’ character

|  |  |  |  |
| --- | --- | --- | --- |
| Bit | Name | Meaning (after add or sub) | How to set |
| N | negative | Result is (-) | MSB of result not counting carry |
| Z | Zero | Result is Zero | Result is zero |
| V | Overflow | Signed Overflow | Result is within bit range |
| C | Carry | Unsigned Overflow | Result is within bit range |

* Pre Index: LDR r1,[r0,#4] : offsets mem address by 4 & stores values of r1 into higher address spaces
* Post Index: LDR r1,[r0],#4 : offsets mem address by 4 & stores value of r1 into lower address spaces
* Pre Index w update: LDR r1,[r0,#4]! : same a pre index but r0 is incremented by 4
* ARMv7 is a modified Harvard architecture
* Horner’s rule: 3x3-7x2+10x-11 can be represented as x(x(3x-7)+10)+11 to reduce # of assembly instructions
* Conditional codes:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| B | BGT | BGE | BLT | BLE | BEQ | BNE |
| Branch always | > | >= | < | <= | = | != |

* B label(branch unconditional): causes a branch to label
* BL label (branch&link): instruction copies the address (PC+4) of next instruction into LR and causes branch to label (instruction is used to call a subroutine)
* BX Rm (Branch indirect): branch to address held in Rm
* BLX Rm (Branch indirect w link): copies address of next instruction into LR and branches to address held in Rm
* BX LR: returns from a subroutine
* Stack is last in-first out, can push & pop data on & off of stack, descending stack is common, descending means when we push to stack, stack occupies lower mem addresses. Full Stack= SP points to last item pushed onto stack, Empty=SP points to next free space on stack
* Compare Instructions: (these instructions update flags based on result and then discard the result)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instruction | Operands | Brief Description | Flags | Operation |
| CMP | Rn, Op2 | Compare | N,Z,C,V | Operand1-operand2 |
| CMN | Rn, Op2 | Compare negative | N,Z,C,V | Operand1+operand2 |
| TEQ | Rn, Op2 | Test equivalence | N,Z,C | Operand1 AND operand2 |
| TST | Rn, Op2 | Test | N,Z,C | Operand1 EOR Operand2 |

* Example (If –Then):
* Example (If-Then-Else):
* Example(For loop)