A minimal hypothetical processor

Word length is 16 bits with the leftmost, most significant bit (MSB) being bit 15. There are eight general registers of 16 bits each. **Register 7 is the program counter (IP)** and Register 6 is the stack pointer (SP). There is also a Processor Status Register/Word (PSW) which indicates the condition code bits (ALU-Flags: Z, C).

The Addressing is linear from memory address 0 through 0xFFFF. All word memory addresses are even. In byte operations, an even address specifies the least-significant byte and an odd address specifies the most-significant byte.

The syntax of the assembler instructions is:

```
instruction source, destination
```

with: destination = destination operation source

examples:

```
MOV R1,R2     ; R2 = R1

ADD R1,R2     ; R2 = R2 + R1

SUB R1,R2     ; R2 = R2 – R1
```

Operand Adressing

Addressing for the Single Operand, Double Operand and Jump instructions is achieved via six bits. They are denominated as *ssssss* (Source) or *dddddd* (Destination) in the instructions.

x	x	x	n	n	n
Mode			Register		

where the modes are as follows: (Reg = Register)

Mode 0	000	Reg direct	Direct addressing of the contents of the register	Rn
Mode 1	001	Reg indirect	Contents of Reg is the address	(Rn)
Mode 2	010	Reg indirect with AutoIncr	Contents of Reg is the address, then Reg incremented	(Rn)+
Mode 3	011	Reg indirect with AutoDecr	Reg is decremented then contents is address	-(Rn)
Mode 4	100	Indexed	Contents of Reg + Following word (X) is address	X(Rn)

Although not special cases, when dealing with **R7** (aka the **IP**), some of these operations are called different things:

x	x	x	1	1	1
Mode			R7		

Mode 2	Immediate	Operand follows the instruction	#N
Mode 5	Absolute, Memory direct	Address of Operand follows the instruction: direct addressing of memory	(A)

Mainstream instructions are broken into Single operand and Double operand instructions, which in turn can be word or byte instructions.

Double Operand Instructions

b	i	i	i	s	s	s	s	s	s	d	d	d	d	d	d
	Opcode			Source						Destination					

Bit 15, b, generally selects between word-sized (b=0) and byte-sized (b=1) operands. In the table below, the mnemonics and names are given in the order b=0/b=1.

The double operand instructions are:

b 000 ssssss dddddd			Non-double-operand instructions.
b 001 ssssss dddddd	MOV/MOVB	*Move Word/Byte*	Moves a value from source to destination.
b 010 ssssss dddddd	CMP/CMPB	*Compare Word/Byte*	Compares values by subtracting the destination from the source, setting the condition codes, and then discarding the result of the subtraction.
b 110 ssssss dddddd	ADD/SUB	*Add/Subtract Word*	Adds the source and destination, storing the results in the destination. Subtracts the source from the destination, storing the results in the destination. Note that this is a special case for b=1, in that it does not indicate that byte-wide operands are used.

Single Operand Instructions

b	0	0	0	i	i	i	i	i	i	d	d	d	d	d	d
	Opcode			Instruction						Destination					

Bit 15, b, generally selects between word-sized (b=0) and byte-sized (b=1) operands. In the table below, the mnemonics and names are given in the order b=0/b=1. Unless otherwise stated, the operand is read for the data to operate on, and the result is then written over that data.

The single operand instructions are:

b 000 101001 dddddd	COM/COMB	*Complement Word/Byte*	Calculates the ones-complement of the operand, and stores it. The ones-complement is formed by inverting each bit (0->1, 1->0) independently.
b 000 010010 dddddd	INC/INCB	*Increment Word/Byte*	Adds one to the destination.
b 000 101011 dddddd	DEC/DECB	*Decrement Word/Byte*	Subtracts one from the destination.
b 000 101101 dddddd	ADC/ADCB	*Add Carry Word/Byte*	Adds the current value of the carry flag to the destination. This is useful for implementing arithmetic subroutines with more than word-sized operands.
b 000 101110 dddddd	SBC/SBCB	*Subtract Carry Word/Byte*	Subtracts the current value of the carry flag from the destination. This is useful for implementing arithmetic subroutines with more than word-sized operands.
b 000 110010 dddddd	ASR/ASRB	*Arithmetic Shift Right Word/Byte*	Shifts the bits of the operand one position to the right. The left-most bit is duplicated. The effect is to perform a signed division by 2.
b 000 110011 dddddd	ASL/ASLB	*Arithmetic Shift Left Word/Byte*	Shifts the bits of the operand one position to the left. The right-most bit is set to zero. The effect is to perform a signed multiplication by 2.
b 000 110101 dddddd	PUSH		Pushes a word onto the current R6 stack from the operand address
b 000 110110 dddddd	POP		Pops a word from the R6 stack to the operand

Branches

b	0	0	0	b	b	b	b	d	d	d	d	d	d	d	d
Branche Code								Destination							

The destination of a branch is +127 to -128 words from the word following the branch instruction itself (relative jump). The sequence of events: the branch instruction is read from memory and the IP incremented. If the branch is to be taken, the offset is then added to the current value of the IP. Since the IP has already been incremented, the offset is thus relative to the following word. Note that all branch instructions are one word long.

The various branches test the values of specific condition codes, and if the tests succeed, the branch is taken. The condition codes are Z (zero) and C (carry).

0 000 0001 ddddddddd	BR	Branch Always
0 000 0010 ddddddddd	BNE,BNZ	Branch if Not Equal (Z==0)
0 000 0011 ddddddddd	BEQ,BZ	Branch if EQual (Z==1)
1 000 0110 ddddddddd	BCC,BNC	Branch if Carry Clear (C == 0)
1 000 0111 ddddddddd	BCS,NC	Branch if Carry Set (C == 1)

Jumps and Calls

0	0	0	0	1	0	0	1	1	1	d	d	d	d	d	d
Opcode							Source			Destination					

0000100111 dddddd	JSR	*Jump to Subroutine*	The actual sequence of steps taken is: MOV IP,-(R6) JMP <destination>

0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1
Opcode													Destination		

0000000010000 ddd	RTS	*ReTurn from Subroutine*	Undoes the effects of a JSR. The actual operations involved are: MOV (R6)+,IP

0	0	0	0	0	0	0	0	1	0	d	d	d	d	d	d
Opcode										Destination					

0000000001 dddddd	JMP	*JuMP*	Loads the destination address into the IP, thus effecting an unconditional jump.