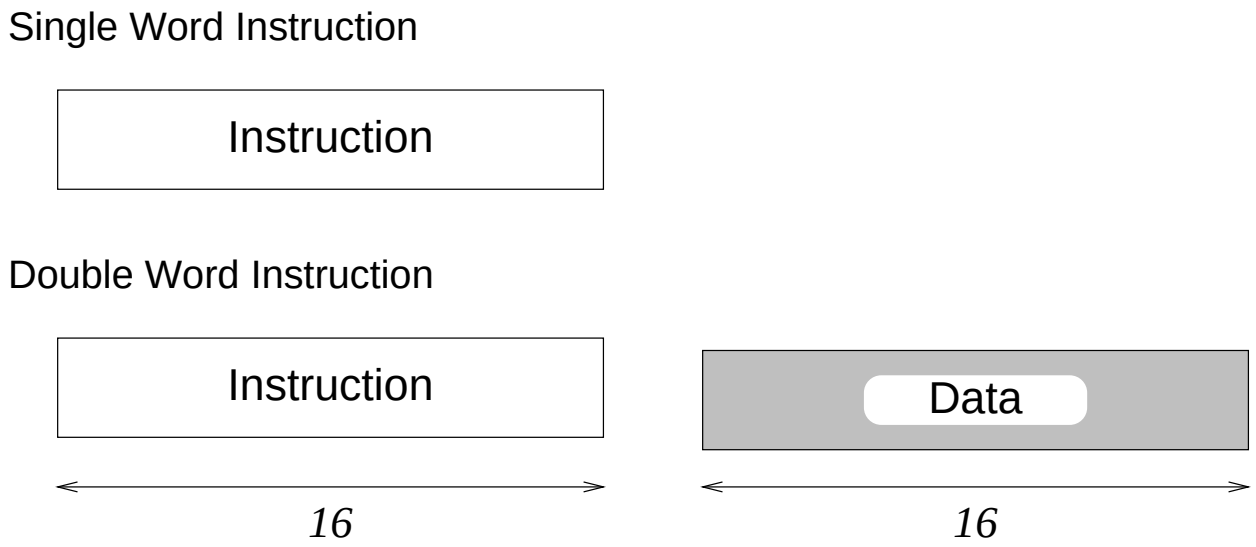
Basic Processor Design – CISC

- Instruction Length is Variable
  - instructions with implied operands will typically be 1 word (i.e. 16 bits)
  - instructions with an explicit operand will be 2 words



The data word contains either the operand itself (immediate addressing mode) or a value which will be used to calculate the operand address in memory.

Basic Processor Design – CISC

- Register-Memory Architecture

  A typical arithmetic or logic instruction will take one operand from a register and one from memory. It will return the result to a register.

  e.g. `ADDA 203` $A \leftarrow A + mem(203)$

  A is the implied operand while 203 is the address of the explicit operand.

- A single instruction may give rise to a complex sequence of events

  e.g. `JSR +137` $SP \leftarrow SP - 1$

  $mem(SP) \leftarrow PC$

  $PC \leftarrow PC + 137$

  The old value of the program counter is stored on a stack in memory during a jump to subroutine instruction.

Basic Processor Design – CISC

$Q1_{CISC}$
How many Data Registers and Address Registers?

The minimum system for this exercise will have just one data register (Accumulator: A) and one address register (Stack Pointer: SP).

- Extra Address Registers.

  Having another address register (e.g. Index Register: X) should help to reduce the number of memory accesses to speed up the processor. It should also help to simplify the programming.

- Extra Data Registers.

  Having another data register (e.g. Accumulator: B) is likely to have less effect unless you add support for Register-Register arithmetic/logic instructions.

- Multi-Purpose Registers.

  For this exercise it may be beneficial to allow any register to be used as either a data register or an address register, with just one of the registers acting also as the stack pointer.

Basic Processor Design – CISC

Q2$_{CISC}$
What Addressing Modes should we support for Arithmetic/Logic/Load & Store?

Basic addressing modes[1]:

Addressing Mode	Assembly Language Syntax	Semantics
Immediate	`ADDA #imm`	$A \leftarrow A + imm$
	—	—
Direct	`ADDA addr`	$A \leftarrow A + mem(addr)$
	`STA addr`	$mem(addr) \leftarrow A$
Indexed with X	`ADDA X,offset`	$A \leftarrow A + mem(X + offset)$
	`STA X,offset`	$mem(X + offset) \leftarrow A$
Indexed with SP	`ADDA SP,offset`	$A \leftarrow A + mem(SP + offset)$
	`STA SP,offset`	$mem(SP + offset) \leftarrow A$
Stack (pop)	`ADDA SP++`	$A \leftarrow A + mem(SP); SP \leftarrow SP - 1$
(push)	`STA --SP`	$SP \leftarrow SP - 1; mem(SP) \leftarrow A$

[1]It is not necessary to implement all of these modes

Basic Processor Design – CISC

Assembly language conventions used here:

`#nnn`	The literal value nnn is the operand
`nnn`	nnn is the address of the operand
`Reg,nnn`	$Reg + nnn$ is the address of the operand
`(nnn)`	nnn is the address of a pointer to the operand
`--Reg`	Reg is pre-decremented before the address is calculated
`Reg++`	Reg is post-incremented after the address is calculated

Less useful addressing modes:

Addressing Mode	Assembly Language Syntax	Semantics
Indirect	`ADDA (point_addr)`	$A \leftarrow A + mem(mem(point\_addr))$
(Pre-)Indexed Indirect	`ADDA (X,offset)`	$A \leftarrow A + mem(mem(X + offset))$
Indirect (Post-)Indexed	`ADDA X,(point_addr)`	$A \leftarrow A + mem(X + mem(point\_addr))$

These modes all use multiple memory accesses to reach data in complex data structures. Since such data structures are unlikely to be required for this processor, the extra complexity involved in implementing them is not justified by the benefit gained.

Basic Processor Design – CISC

$Q3_{CISC}$
What Instructions will we support?

Because we do not need to include an immediate field in the 16 bit instruction word, we can have more instructions[2].
This is one possible set based on the CISC example processor model in Verilog:

Type	Function	Mnemonic	Function	Mnemonic
Arithmetic	Add	ADDA	Subtract	SUBA
	Add with Carry	ADCA	Subtract with Borrow	SBCA
Logic	Bitwise NOT	COMA	Bitwise AND	ANDA
	Bitwise OR	ORA	Bitwise Exclusive OR	XORA
	Logical Shift Right	LSRA	Logical Shift Left	LSLA
Data Movement	Load	LDA,LDX,LDS	Store	STA,STX,STS
Control Transfer	Branch Always	BA		
	Branch if Equal	BEQ	Branch if Not Equal	BNE
	Branch to Subroutine	BSR	Return from Subroutine	RTS

[2]this lack of constraint is all the more reason to choose the instruction set carefully

$Q4_{CISC}$
Which Addressing Modes are Supported for each Instruction?

- Inherent

  For certain instructions (LSRA, LSLA, COMA, RTS) the addressing mode is implied/inherent in the instruction. These are always single word instructions.

- PC Relative

  For Branch instructions the addressing mode is PC-relative (some processors use PC-absolute instead). The branch destination is calculated by adding the branch offset to the current program counter (PC).

- Other Instructions

  The remaining instructions can support any of the addressing modes described in $Q2_{CISC}$[3].

[3] with the exception that a store instruction in immediate mode is meaningless

$Q5_{CISC}$
How is the Instruction Coded?

Normally CISC instructions are compactly coded leading to complex decode logic. The most one might expect is an instruction field that specifies the addressing mode.

With a word length of 16 bits and no immediate to include we can produce an orthogonal coding with many sub-fields for easy decoding (and potentially more power).

Note: just a few of the potential sub-fields are shown here.