Addressing Modes

1 CPU Fetch Execute Cycle

1. CPU sends the address in the program counter (PC) to the Memory Address Register (MAR)
2. Increment the PC
3. Get the instruction identified by the MAR into the Memory Data Register (MDR)
4. Move the instruction from the MDR to the instruction register
5. Move the instruction from the IR to the control unit for **decoding**
    - Send the operation to the ALU
    - Put the address if data to be operated upon in a register
6. Send the address if data to be operated upon in a register
7. Read the data from memory into the MDR
8. Move the data from the MDR to an accumulator into the ALU
9. Do the operation and store the result in an accumulator in the ALU

2 Addressing

Sometimes we would like to:

- Address a large amount of memory with only a few bits
- Use indexes to loop or examine a table or array
- Relocate data or programs in the memory
- Operate on the registers rather than actual memory

This can be achieved using alternative addressing modes

Sometimes we don't know where exactly we want to index to, but instead know the relative position to where you currently are.

2.1 Alternatives to direct addressing

- **Direct addressing** - the address read in the instruction is the address of the actual data
- **Immediate addressing** - the address read in the instruction **is the data** to be used. Suitable for constants, e.g. add 1
- **Indirect addressing** - the address read in the instruction is the address of a memory location containing the address of the actual data
- **Register indirect addressing** - the address read in the instruction is the address of a register containing the address of the actual data
- **Indexed addressing** - the address read in the instruction should have an index value (contained in some register) added to obtain the address of the data

3 Adding a column

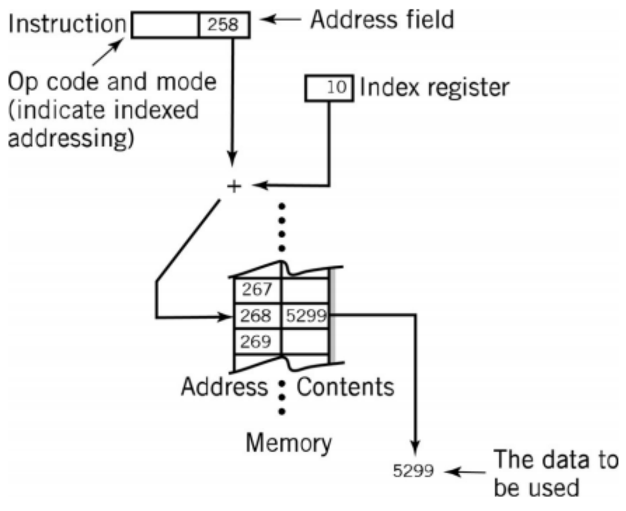
3.1 Direct

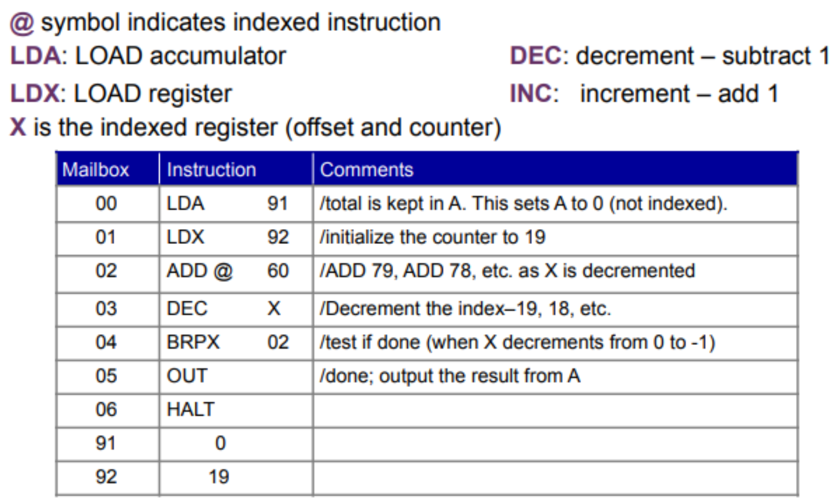
- The instruction in a specific address is changed during the program
- This is called **impure code** - pure code does not change when it is run
- It can cause difficulties:
  - If the index is not reset, next time the program will run differently
  - If the program is interrupted the address will not be reset
  - The program cannot be run from ROM

3.2 Indirect

Loop over a table, and increment a value each time

3.3 Indexed





@ symbol indicates indexed instruction
LDA: LOAD accumulator
LDX: LOAD register
X is the indexed register (offset and counter)
DEC: decrement – subtract 1
INC: increment – add 1

Mailbox	Instruction		Comments
00	LDA	91	/total is kept in A. This sets A to 0 (not indexed).
01	LDX	92	/initialize the counter to 19
02	ADD @	60	/ADD 79, ADD 78, etc. as X is decremented
03	DEC	X	/Decrement the index–19, 18, etc.
04	BRPX	02	/test if done (when X decrements from 0 to -1)
05	OUT		/done; output the result from A
06	HALT		
91	0		
92	19		

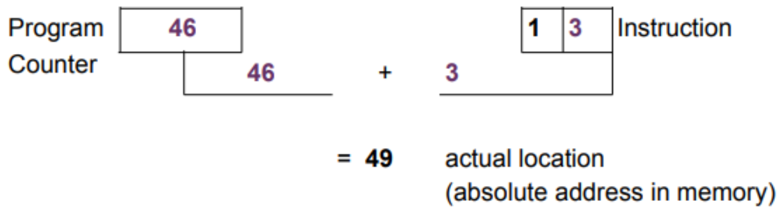
Note the index is decremented each time

4 Addressing mode

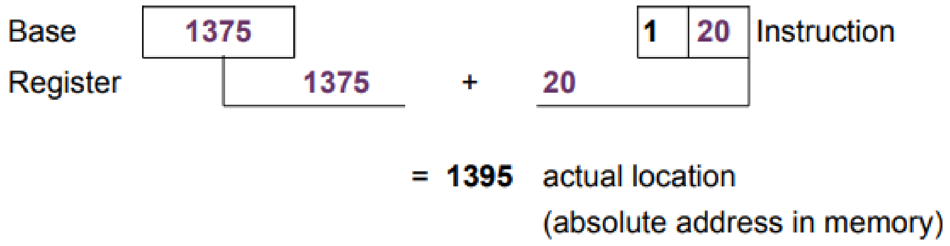
We used * and @ to indicate addressing mode
In a real CPU, the **instruction itself** will need to contain the addressing mode

5 Alternative to absolute addressing

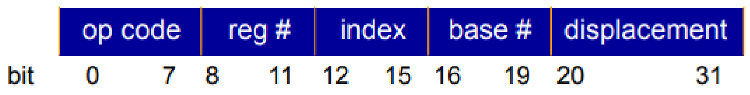
Absolute - The address read is the one the minion should go to (except for indexed addressing)
Relative addressing - The address read is an offset to the current instruction address

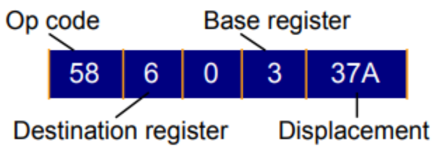


Base offset addressing - the address read is offset by the current value in a special 'base' register



IBM zSystem Load instruction:





6 Instructions

6.1 Data movement

Movement of data between:

- Registers
- Registers and memory
- Memory locations (infrequently)

The most heavily used part of an instruction set and therefore it must be the most flexible, using different sizes of data

6.2 Arithmetic Instructions:

Most modern CPU's instruction set includes:

- Integer addition and subtraction
- Integer multiplication and division
- Floating point instructions

6.3 Program control:

- JUMPS
- BRANCHES (Conditional or Unconditional)
- CALL
- RETURN

6.4 Boolean logic instructions:

- **Bit manipulation**
  - Allows programmers to control program flow by providing the mechanism for them to design their own 'flags'
  - Instructions include set and test
- **Shift and Rotate Instructions**
  - Shift: Move data bits left or right one or more bits at a time. Bits shifted out of the end may trigger a flag or drop off the end. Fill in with 0s
  - Rotate: move data bits left or right one of more bits at a time. Bits shifted to the end are rotated back to the beginning. Loop round

6.5 Single operand manipulation

- Negating a value
- Incrementing a value
- Decrementing a value
- Setting a register to zero

6.6 Multiple data instructions

6.7 Stack instructions

- POP
- PUSH