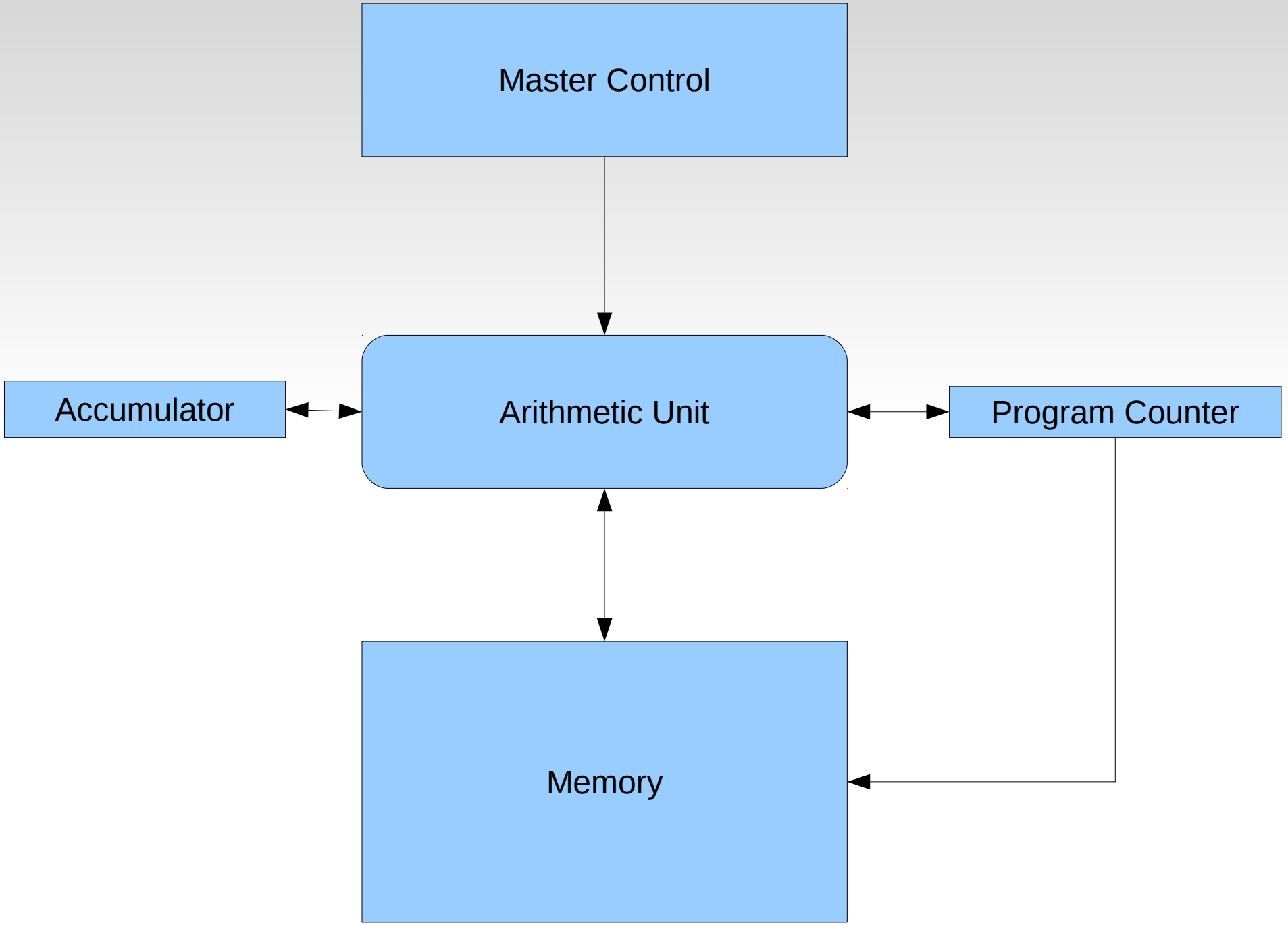
How A Computer Works

- This is a simplified picture of how a computer works.
- It is quite like many early computers, but modern computers have many variations on the theme.


Master Control
Accumulator
Arithmetic Unit
Program Counter
Memory

The Instruction Set

- Accumulator
  - Clear Accumulator
  - Load Accumulator
  - Load Accumulator Immediate
  - Store Accumulator
- Arithmetic
  - Add to Accumulator
  - Add to Accumulator Immediate
    - Ditto Subtract, Multiply, Divide.

The Instruction Set (cont)

- Branching
  Jump to address
  Jump if Accumulator Zero
    Ditto non-zero, >= zero, < zero
- Procedure Call
  For our first example, we will just assume
    call print
  which prints the contents of the accumlator.
- Stopping

Instruction Mnemonics

- CLA
- LDA <memory address>
- LDAI <constant>
- STA <memory address>
- ADD <memory address>
- ADDI <constant>
- SUB <memory address>
- SUBI <constant>
- MUL <memory address>
- MULI <constant>

Instruction Mnemonics (cont)

- `DIV  <memory address>`
- `DIVI  <constant>`
- `JMP  <memory address>`
- `JNE  <memory address>`
- `JEQ  <memory address>`
- `JGE  <memory address>`
- `JLT  <memory address>`
- `CALL  print`
- `HLT`

- It turns out that this instruction set is enough to write ANY possible program. (In fact, in theory you do not even need most of them).
- Let us see how a small program looks. We calculate some Fibonacci numbers:

  1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

  Each new number is the sum of the previous two.

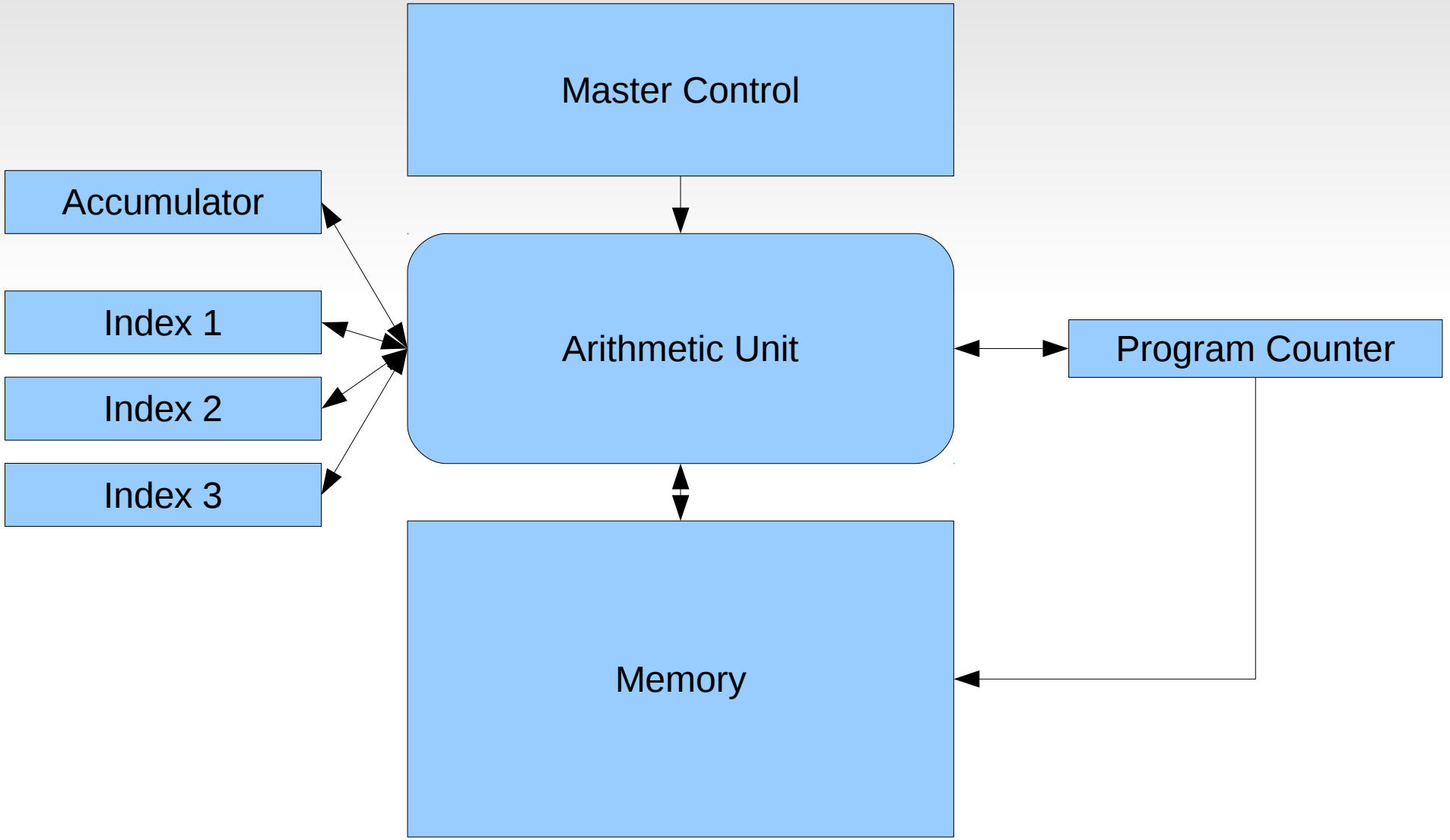
- All memory addresses are in fact numbers (starting with 0) and even the instructions are just numbers, with part of the word representing the operation – 5 bits would allow 32 instructions – and the rest the address or constant.
- But it is useful to use the mnemonics for instructions, and to give names to the important memory locations. An ASSEMBLER is a program that will make the conversion automatically.

Location	Operation	Operand
0	JMP	Start
Prev:		3
Curt:		5
Next:		0
Limit:		40
Start:	LDA	Prev
	ADD	Curt
	STA	Next
	CALL	print
	LDA	Curt
	STA	Prev
	LDA	Next
	STA	Curt
	SUB	Limit
	JLT	Start
	HLT	

Extending The Instructions

- Our computer can handle arrays, for example, because it could load an instruction, add 1 to the address portion, and store it back again. But this is considered bad form.
- Usually, we would add an extra register (or three) to modify addresses. These registers typically can be loaded, stored either from memory or the accumulator and have add and subtract operations only.

Indexed Instructions



Indexed Instructions

- LDA <memory address>,1
- STA <memory address>,2
- MUL <memory address>,3
- ENI <value>,1
- LDI <memory address>,1
- STI <memory address>,3
- INCI <constant>,2
- DECI <constant>,1
- TAI 1
- TIA 2

Indexed Instructions

- The index registers add several new instructions, and some of the existing ones need an additional 2 bits to indicate which index register to use (with 0 indicating none).
- They certainly make using arrays much simpler.

Program With Indexing

```
0       ENI     Data,1
        LDA     0,1
        STA     Best        ; Best so far
Loop    INCI    1,1
        LDA     0,1         ; Get next
        JEQ     Done
        SUB     Best
        JLT     Loop
        LDA     0,1         ; New best
        STA     Best
        JMP     Loop
Done    LDA     Best
        CALL    print
        HLT
Best    
```

```
Data            1
                5
                2
                7
                8
                3
                2
                0       : Marks end
```

- Find and print the largest of a list of positive numbers

The CALL Instruction

- Obviously it has to store a return address
  (current PC + 1)
  to be able to get back. There are several ways to do it.
  - In the target address as a JMP instruction, then increment the PC to get the first instruction of the procedure.
  - In a register. Only useful if the machine has multiple registers
  - In a memory stack. Most modern computers do this.

Stacks

- Think of a stack as a pile of magazines.
  - You can easily access the top one.
  - You could access the next few under it
  - Much harder to get to the bottom ones.
- A new magazine can only be added at the top.
- Only the top magazine can be removed - LIFO.
- In a computer, stacks usually are upside-down – they grow downwards in memory

- Most procedural languages (Algol, Pascal, C) allow a subroutine to call itself recursively. The first two options do not easily support this.
- Suppose we designate index register 3 as a ”stack pointer” (SP) We point it to the end of a suitable block of memory. The CALL instruction procedes:
  - Increment the PC (as usual)
  - Decrement the SP.
  - Store the PC at the address pointed to by the SP
  - Place the destination address in the PC

- To resume at the end of the CALL, we need a RETURN instruction.
  - Load the PC from the memory pointed to by the SP
  - Increment the SP.
- In addition, the stack can be used to store parameter values. We add some useful instructions:
  - PUSH Equivalent to

```
DECI  1,SP
STA   0,SP
```

  - POP Equivalent to

```
LDA   0,SP
INCI  1,SP
```

- We might also want to be able to push and pop index registers.
- Here is a procedure call to calculate the next Fibonnaci number. It is called by:
- Note how we push two parameters to the stack, and clean up the stack afterwards.

Location	Operation	Operand
	LDA	Prev
	PUSH	
	LDA	Curt
	PUSH	
	CALL	NextFib
	INCI	2,SP

Location	Operation	Operand	
NextFib	LDA	2,SP	Prev
	ADD	1,SP	Curt
	RETURN		

- Note how we can access the two parameters.
- The result is returned in the accumulator (we could store it in the stack – the caller would have to make space – or put a pointer to where we want the result to go in the stack).

- We glossed over how this would fit in the program to calculate Fibonacci numbers. It might look like this.

Locaton	Operation	Operand
0	JMP	Start
Prev:		3
Curt:		5
Limit:		40
Start:	LDA	Prev
	PUSH	
	LDA	Curt
	PUSH	
	STA	Prev
	CALL	NextFib
	STA	Curt
	CALL	print
	LDA	Curt
	SUB	Limit
	JLT	Start
	HLT	

Summary

- A simple computer consists of control unit, arithmentic unit, registers and memory.
- Many modern computers are much more complex
  - e.g. multiple aithmetic units, automatic memory caches.
- Most have a relatively simple instruction set.
- Programmers use compilers to generate thousands of instructions from the more power ful commands of modern programming languages.