Target

Our input is a binary .gb file which we are going to run in C. The binary is going to be converted to opcodes based off of the Z80 structure that the original gameboy was based off of. Our C code will run the opcodes on emulated hardware that we will construct. The running emulator should be able to process the video, controls, and engine running in the background.

Input Language

Two Types:

Opcodes and operands (register, memory addresses, and immediate values)

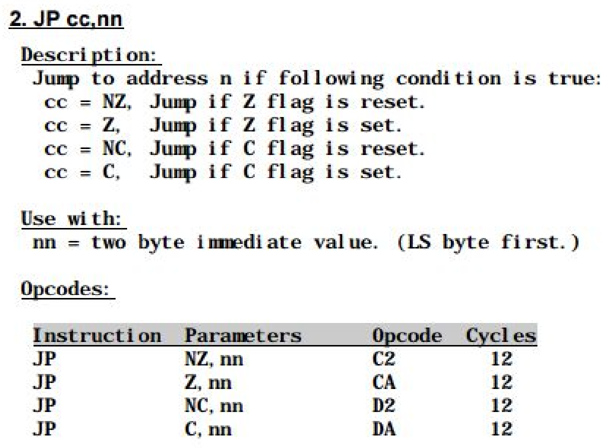
Related Operators:

LD, ADD, ADC, etc. There are about 200 of these with different combinations of arguments.

Condition Construct:

The Z80 instruction set uses the zero and carry flags in-order to implement a condition constructor. These flags can be manipulated by various operators in the language.

JP cc, nn



2. JP cc,nn

Description:
Jump to address n if following condition is true:
cc = NZ, Jump if Z flag is reset.
cc = Z, Jump if Z flag is set.
cc = NC, Jump if C flag is reset.
cc = C, Jump if C flag is set.

Use with:
nn = two byte immediate value. (LS byte first.)

Opcodes:

Instruction	Parameters	Opcode	Cycles
JP	NZ, nn	C2	12
JP	Z, nn	CA	12
JP	NC, nn	D2	12
JP	C, nn	DA	12

JP cc, n (n is the amount of bytes to jump)
CALL cc, nn
RET cc
http://marc.rawer.de/Gameboy/Docs/GBCPUman.pdf

Looping Construct:

Using the condition construct you can create a looping structure using the carry or zero flags.

Procedure Construct:

Blocks of code are essentially stuck in between jump statements, some jumps may make code start executing in the middle of a block.

JP is the important code for identifying procedures.

Lexical Spec

Our lexical spec is our reader file. Included is the original C code as well as it compiled. Included is a readme for more information.

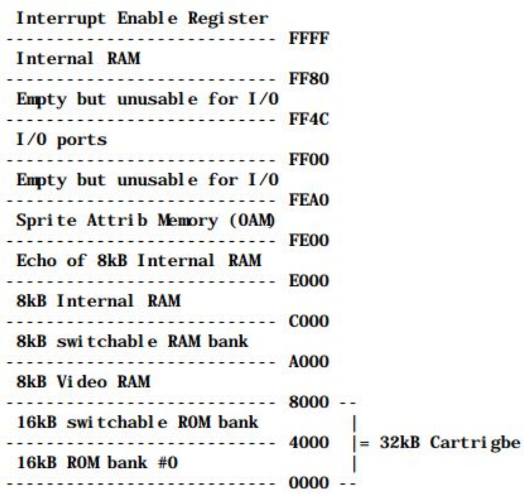
The Header

Each gb rom starts with a 336 byte header, the first 256 bytes are various restart and start addresses that we won't go into now. The next 80 bytes includes the execution point (which usually points to the code immediately after the header), a nintendo graphic, and various meta data pertaining to this particular rom including the license, region, cartridge type, ram size, rom size, the title of the game, and lastly a checksum that is run on boot.

Depending on the start address given in the header the following bytes includes the main program. This code is written in machine code with the Z80's instruction set. The various instructions and the input format in addition to the register format are given on the last page.

How the game boy uses the rom

Once the initial Game Boy log flashes across your screen and the jump call to the main program is executed there will be a bunch of various load calls into the memory of the GameBoy. Each load call into the various points in this memory have different meanings. Here is a general overview of what the gameboy memory map looks like:



```
 Interrupt Enable Register
--------------------------- FFFF
 Internal RAM
--------------------------- FF80
 Empty but unusable for I/O
--------------------------- FF4C
 I/O ports
--------------------------- FF00
 Empty but unusable for I/O
--------------------------- FEA0
 Sprite Attrib Memory (OAM)
--------------------------- FE00
 Echo of 8kB Internal RAM
--------------------------- E000
 8kB Internal RAM
--------------------------- C000
 8kB switchable RAM bank
--------------------------- A000
 8kB Video RAM
--------------------------- 8000 --
 16kB switchable ROM bank         |
--------------------------- 4000  |= 32kB Cartrigbe
 16kB ROM bank #0                 |
--------------------------- 0000 --
```

http://marc.rawer.de/Gameboy/Docs/GBCPUman.pdf

Some of this memory is mapped out to specific hardware pieces. By reading and writing to those registers, audio and video is played. Since our output is an emulator, the hardware memory mapping and the actual hardware emulation will have to be incorporated into the final project.

The gameboy also has a flag register, 8 8 bit registers which can be combined into 4 16 bit registers which are used for instruction operations as well as 2 special 16 bit registers used for the program counter and stack pointer.