PowerPC Lessons

By: Bad Luck Brian
Powered by: Bad Luck Modders

Who are Bad Luck Modders [BLM*] ?

We are modders who mainly wants to teach anyone who is willing to learn about modding, Programming and reversing. We only have a Skype chat as of now and getting famous is not a priority nor is modding online.

Side note: I've learned most of what I know about PPC (PowerPC) using IDA and the Prodg debugger to understand what most of the instructions were doing and used IBM's site and Google to understand their meanings. Of course I might make some mistakes but the most important part will be there.

Table Of Contents

Requirements

Note: (Red: It's a must | Orange: Recommended | Green: Optional)
Note2: These requirements are made so you can start modding after this tutorial.

1. COMMON SENSE (in other terms, use your brain..)
2. IDA PRO ADVANCED(With the PS3 plugins)
3. SN Systems tools (Prodg debugger and Target Manager)
4. A PS3 on DEX
5. Visual Studio | Or anything similar to compile your projects
6. Some programming skills
7. HxD Editor
8. A good understanding of how a computer works
9. Notepad++ (really useful)

I also recommend viewing this PDF in Google Chrome and using the zoom feature to view the images.

Some of the required files are included at the following link:
https://anonfiles.com/file/4bcbd74e16c327403cdb12e136010479

Setting up your PC (1/5)

NOTE: Everything in this tutorial will be made on Windows 8 using the update 1.13 of Call of Duty Ghosts. (It works with windows 7 too).

First of all you will need to install everything from this package.
-Sn systems tools
-IDA PRO (you don't need to install it's portable)
-Visual studio
-HxD Editor
-Notepad++

This will require you to use your brain. I really hope you can install a few softwares without the need to be assisted. Just remember where you installed SN System and IDA.

NOTE: In IDA Pro there is a file named 'ida6x.dll' or something similar and it might show up as a virus or something. Just exclude it from your anti-virus or do everything in a virtual machine. Remember it's a cracked version.

My directories:

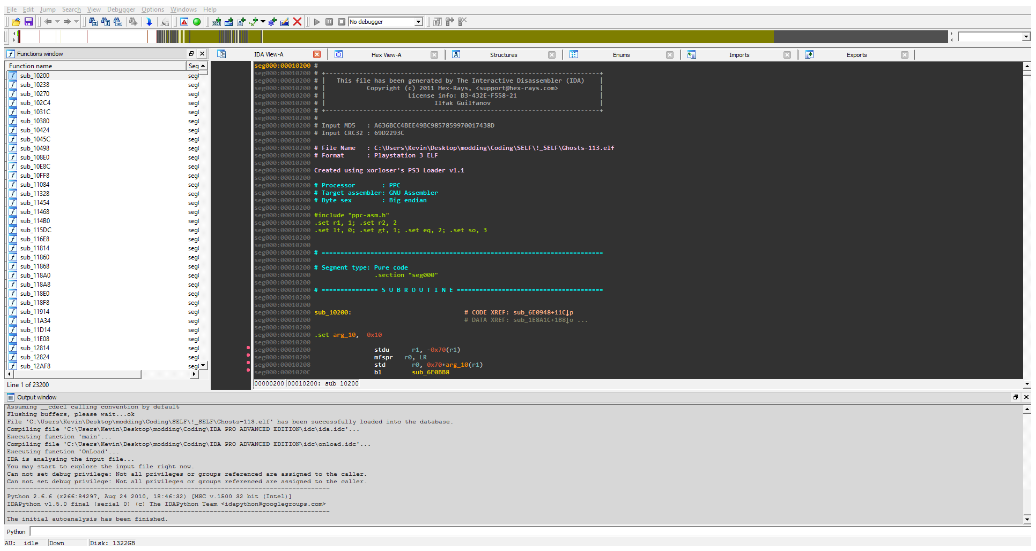
SN Systems:
C:\Program Files (x86)\SN Systems

IDA:
C:\Users\BLB\Desktop\modding\Coding\IDA

Setting up your PC (2/5)

Now that everything is installed we want to make it a bit more simple and faster to use.

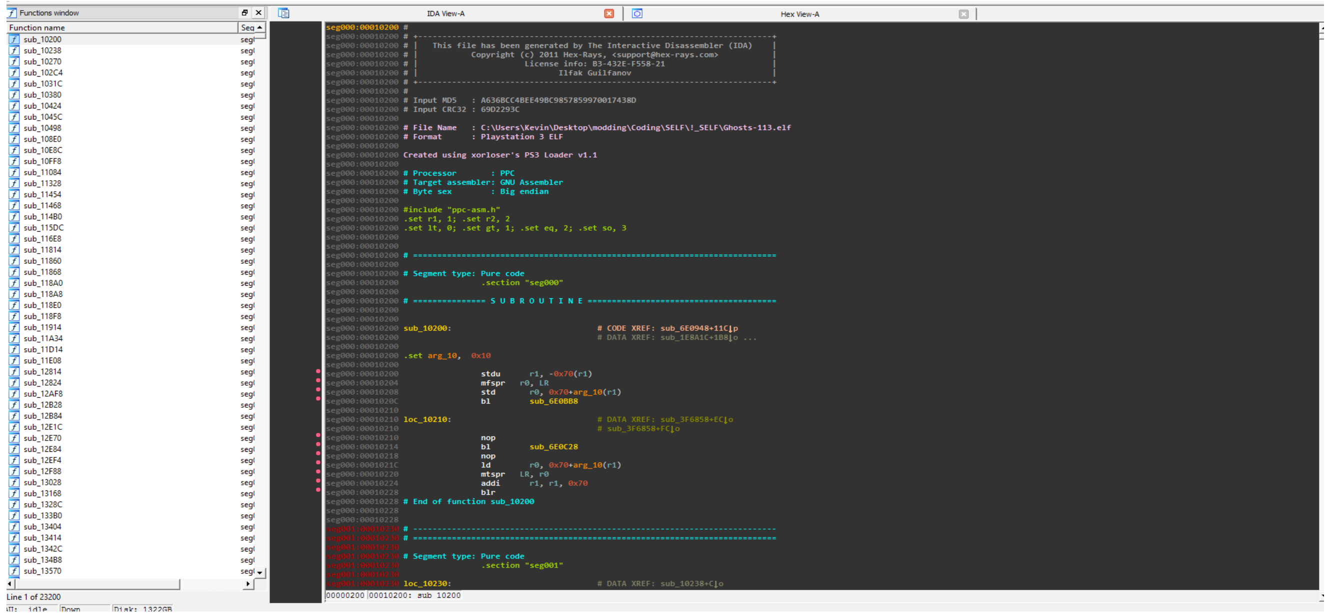
Get an .elf file and right click on it, then select properties.
Click on 'Open with' and select 'idaq.exe' in your IDA folder.
This way you will be able to open .elf files directly with IDA.



```
seg000:00010200 #
seg000:00010200 # +-------------------------------------------------------------------------+
seg000:00010200 # |   This file has been generated by The Interactive Disassembler (IDA)    |
seg000:00010200 # |           Copyright (c) 2011 Hex-Rays, <support@hex-rays.com>           |
seg000:00010200 # |                      License info: B3-432E-F558-21                      |
seg000:00010200 # |                             Ilfak Guilfanov                             |
seg000:00010200 # +-------------------------------------------------------------------------+
seg000:00010200 #
seg000:00010200 # Input MD5    : A636BCC4BEE49BC9857859970017438D
seg000:00010200 # Input CRC32 : 69D2293C
seg000:00010200
seg000:00010200 # File Name   : C:\Users\Kevin\Desktop\modding\Coding\SELF\!_SELF\Ghosts-113.elf
seg000:00010200 # Format      : Playstation 3 ELF
seg000:00010200
seg000:00010200 Created using xorloser's PS3 Loader v1.1
seg000:00010200
seg000:00010200 # Processor        : PPC
seg000:00010200 # Target assembler: GNU Assembler
seg000:00010200 # Byte sex         : Big endian
seg000:00010200
seg000:00010200 #include "ppc-asm.h"
seg000:00010200 .set r1, 1; .set r2, 2
seg000:00010200 .set lt, 0; .set gt, 1; .set eq, 2; .set so, 3
seg000:00010200
seg000:00010200
seg000:00010200 # ===========================================================================
seg000:00010200
seg000:00010200 # Segment type: Pure code
seg000:00010200                 .section "seg000"
seg000:00010200
seg000:00010200 # =============== S U B R O U T I N E =======================================
seg000:00010200
seg000:00010200
seg000:00010200 sub_10200:                              # CODE XREF: sub_6E0948+11C↓p
seg000:00010200                                         # DATA XREF: sub_1E8A1C+1B8↓o ...
seg000:00010200
seg000:00010200 .set arg_10,  0x10
seg000:00010200
seg000:00010200                 stdu      r1, -0x70(r1)
seg000:00010204                 mfspr   r0, LR
seg000:00010208                 std       r0, 0x70+arg_10(r1)
seg000:0001020C                 bl        sub_6E0BB8
00000200 00010200: sub 10200
```

```
Assuming __cdecl calling convention by default
Flushing buffers, please wait...ok
File 'C:\Users\Kevin\Desktop\modding\Coding\SELF\!_SELF\Ghosts-113.elf' has been successfully loaded into the database.
Compiling file 'C:\Users\Kevin\Desktop\modding\Coding\IDA PRO ADVANCED EDITION\idc\ida.idc'...
Executing function 'main'...
Compiling file 'C:\Users\Kevin\Desktop\modding\Coding\IDA PRO ADVANCED EDITION\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Can not set debug privilege: Not all privileges or groups referenced are assigned to the caller.
Can not set debug privilege: Not all privileges or groups referenced are assigned to the caller.
---------------------------------------------------------------------------------------
Python 2.6.6 (r266:84297, Aug 24 2010, 18:46:32) [MSC v.1500 32 bit (Intel)]
IDAPython v1.5.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
---------------------------------------------------------------------------------------
The initial autoanalysis has been finished.
```

Now open IDA and it should look like this(Don't mind the theme):
Close every tabs except IDA-view and Hex-View so it will look like this:



```
Function name   Seg
sub_10200       seg
sub_10238       seg
sub_10270       seg
sub_102C4       seg
sub_1031C       seg
sub_10380       seg
sub_10424       seg
sub_1045C       seg
sub_10498       seg
sub_108E0       seg
sub_10E8C       seg
sub_10FF8       seg
sub_11084       seg
sub_11328       seg
sub_11454       seg
sub_11468       seg
sub_114B0       seg
sub_115DC       seg
sub_116E8       seg
sub_11814       seg
sub_11860       seg
sub_11868       seg
sub_118A0       seg
sub_118A8       seg
sub_118E0       seg
sub_118F8       seg
sub_11914       seg
sub_11A34       seg
sub_11D14       seg
sub_11E08       seg
sub_12814       seg
sub_12824       seg
sub_12AF8       seg
sub_12B28       seg
sub_12B84       seg
sub_12E1C       seg
sub_12E70       seg
sub_12E84       seg
sub_12EF4       seg
sub_12F88       seg
sub_13028       seg
sub_13168       seg
sub_1328C       seg
sub_133B0       seg
sub_13404       seg
sub_13414       seg
sub_1342C       seg
sub_134B8       seg
sub_13570       seg
Line 1 of 23200
AU:  idle   Down   Disk: 1322GB
```

```
seg000:00010200 #
seg000:00010200 # +-------------------------------------------------------------------------+
seg000:00010200 # |   This file has been generated by The Interactive Disassembler (IDA)    |
seg000:00010200 # |           Copyright (c) 2011 Hex-Rays, <support@hex-rays.com>           |
seg000:00010200 # |                      License info: B3-432E-F558-21                      |
seg000:00010200 # |                             Ilfak Guilfanov                             |
seg000:00010200 # +-------------------------------------------------------------------------+
seg000:00010200 #
seg000:00010200 # Input MD5    : A636BCC4BEE49BC9857859970017438D
seg000:00010200 # Input CRC32 : 69D2293C
seg000:00010200
seg000:00010200 # File Name   : C:\Users\Kevin\Desktop\modding\Coding\SELF\!_SELF\Ghosts-113.elf
seg000:00010200 # Format      : Playstation 3 ELF
seg000:00010200
seg000:00010200 Created using xorloser's PS3 Loader v1.1
seg000:00010200
seg000:00010200 # Processor        : PPC
seg000:00010200 # Target assembler: GNU Assembler
seg000:00010200 # Byte sex         : Big endian
seg000:00010200
seg000:00010200 #include "ppc-asm.h"
seg000:00010200 .set r1, 1; .set r2, 2
seg000:00010200 .set lt, 0; .set gt, 1; .set eq, 2; .set so, 3
seg000:00010200
seg000:00010200
seg000:00010200 # ===========================================================================
seg000:00010200
seg000:00010200 # Segment type: Pure code
seg000:00010200                 .section "seg000"
seg000:00010200
seg000:00010200 # =============== S U B R O U T I N E =======================================
seg000:00010200
seg000:00010200
seg000:00010200 sub_10200:                              # CODE XREF: sub_6E0948+11C↓p
seg000:00010200                                         # DATA XREF: sub_1E8A1C+1B8↓o ...
seg000:00010200
seg000:00010200 .set arg_10,  0x10
seg000:00010200
seg000:00010200                 stdu      r1, -0x70(r1)
seg000:00010204                 mfspr   r0, LR
seg000:00010208                 std       r0, 0x70+arg_10(r1)
seg000:0001020C                 bl        sub_6E0BB8
seg000:00010210
seg000:00010210 loc_10210:                              # DATA XREF: sub_3F6858+EC↓o
seg000:00010210                                         # sub_3F6858+FC↓o
seg000:00010210                 nop
seg000:00010214                 bl        sub_6E0C28
seg000:00010218                 nop
seg000:0001021C                 ld        r0, 0x70+arg_10(r1)
seg000:00010220                 mtspr   LR, r0
seg000:00010224                 addi      r1, r1, 0x70
seg000:00010228                 blr
seg000:00010228 # End of function sub_10200
seg000:00010228
seg000:00010228
seg001:00010230 # ---------------------------------------------------------------------------
seg001:00010230 # ===========================================================================
seg001:00010230
seg001:00010230 # Segment type: Pure code
seg001:00010230                 .section "seg001"
seg001:00010230
seg001:00010230 loc_10230:                              # DATA XREF: sub_10238+C↓o
00000200 00010200: sub 10200
```

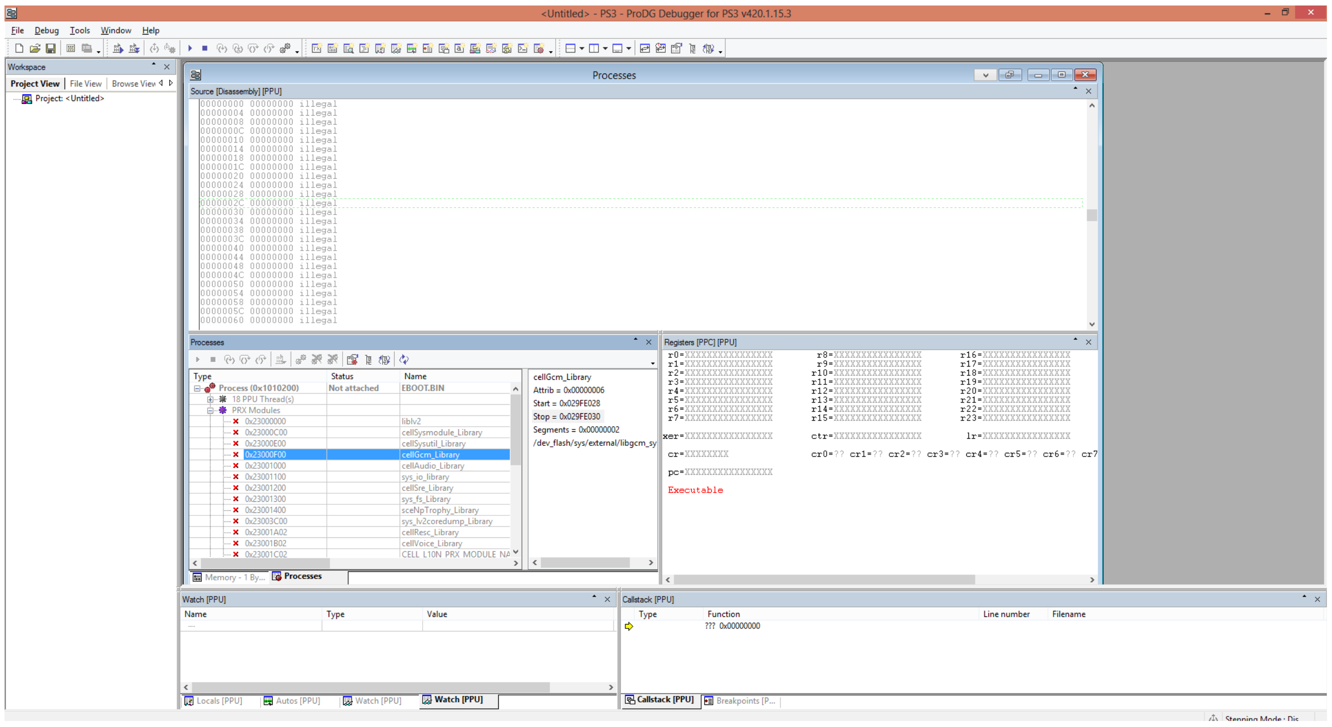
Setting up your PC (3/5)

Now click on Windows > Save Desktop > Check 'default' and press OK.
Now you can close IDA.
Now it's time to make a Modding folder/partition.

Personally i've made a dedicated partition for modding using diskmgmt.
Google on how to make a partition. Else you can just make a folder.

Now go in your SN Systems files. It should be "C:\Program Files (x86)\SN Systems" by default. Then go in the PS3\bin folder and copy the ps3tmapi.dll and paste it into your modding folder. This way it will be easier to get it when needed to make files.

Now it's time to set up your PRODG debugger correctly. Load a game using a debug eboot and open up target manager. Add your target PS3 and open PRODG debugger, it should look like this.



<Untitled> - PS3 - ProDG Debugger for PS3 v420.1.15.3
File Debug Tools Window Help
Workspace
Project View | File View | Browse View
Project: <Untitled>
Processes
Source [Disassembly] [PPU]

```
00000000 00000000 illegal
00000004 00000000 illegal
00000008 00000000 illegal
0000000C 00000000 illegal
00000010 00000000 illegal
00000014 00000000 illegal
00000018 00000000 illegal
0000001C 00000000 illegal
00000020 00000000 illegal
00000024 00000000 illegal
00000028 00000000 illegal
0000002C 00000000 illegal
00000030 00000000 illegal
00000034 00000000 illegal
00000038 00000000 illegal
0000003C 00000000 illegal
00000040 00000000 illegal
00000044 00000000 illegal
00000048 00000000 illegal
0000004C 00000000 illegal
00000050 00000000 illegal
00000054 00000000 illegal
00000058 00000000 illegal
0000005C 00000000 illegal
00000060 00000000 illegal
```

Processes

Type	Status	Name
Process (0x1010200)	Not attached	EBOOT.BIN
18 PPU Thread(s)		
PRX Modules		
0x23000000		liblv2
0x23000C00		cellSysmodule_Library
0x23000E00		cellSysutil_Library
0x23000F00		cellGcm_Library
0x23001000		cellAudio_Library
0x23001100		sys_io_library
0x23001200		cellSre_Library
0x23001300		sys_fs_Library
0x23001400		sceNpTrophy_Library
0x23003C00		sys_lv2coredump_Library
0x23001A02		cellResc_Library
0x23001B02		cellVoice_Library
0x23001C02		CELL_L10N_PRX_MODULE_NA

```
cellGcm_Library
Attrib = 0x00000006
Start = 0x029FE028
Stop = 0x029FE030
Segments = 0x00000002
/dev_flash/sys/external/libgcm_sy
```

Registers [PPC] [PPU]

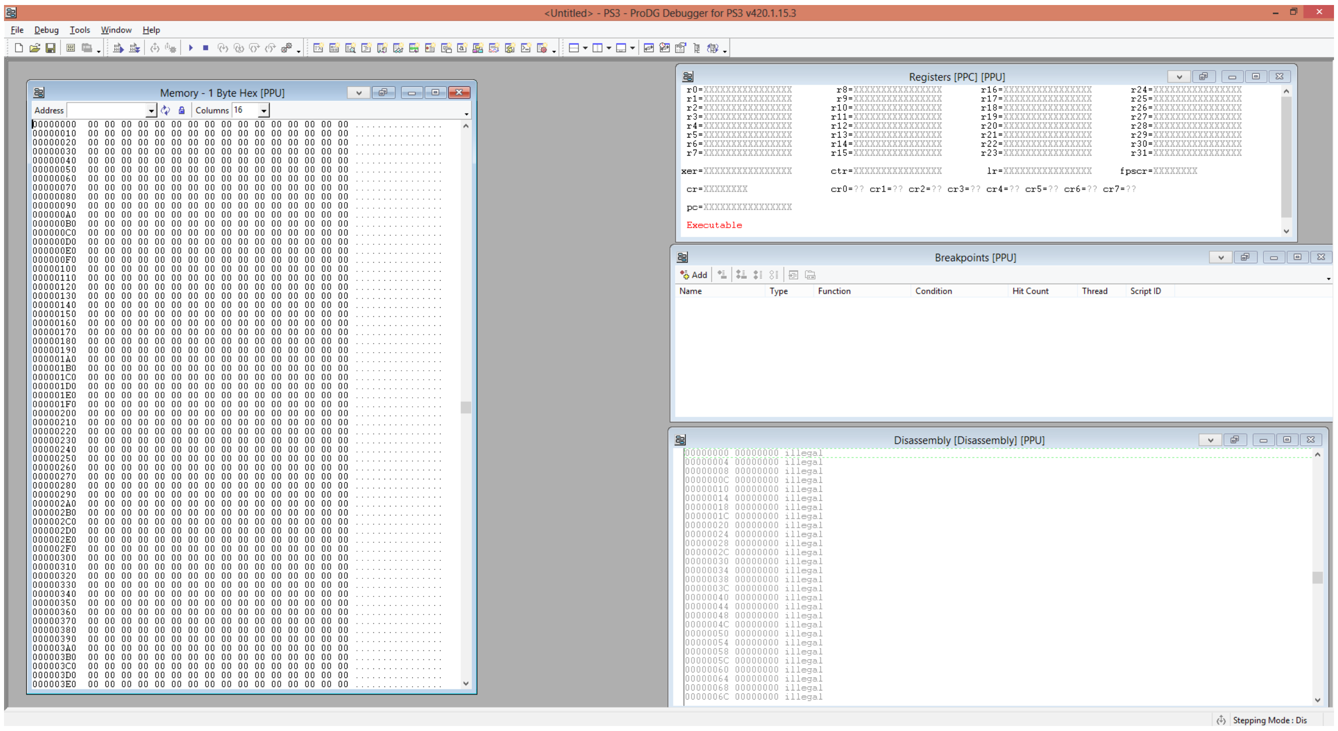
```
 r0=XXXXXXXXXXXXXXXX      r8=XXXXXXXXXXXXXXXX     r16=XXXXXXXXXXXXXXXX
 r1=XXXXXXXXXXXXXXXX      r9=XXXXXXXXXXXXXXXX     r17=XXXXXXXXXXXXXXXX
 r2=XXXXXXXXXXXXXXXX     r10=XXXXXXXXXXXXXXXX     r18=XXXXXXXXXXXXXXXX
 r3=XXXXXXXXXXXXXXXX     r11=XXXXXXXXXXXXXXXX     r19=XXXXXXXXXXXXXXXX
 r4=XXXXXXXXXXXXXXXX     r12=XXXXXXXXXXXXXXXX     r20=XXXXXXXXXXXXXXXX
 r5=XXXXXXXXXXXXXXXX     r13=XXXXXXXXXXXXXXXX     r21=XXXXXXXXXXXXXXXX
 r6=XXXXXXXXXXXXXXXX     r14=XXXXXXXXXXXXXXXX     r22=XXXXXXXXXXXXXXXX
 r7=XXXXXXXXXXXXXXXX     r15=XXXXXXXXXXXXXXXX     r23=XXXXXXXXXXXXXXXX

xer=XXXXXXXXXXXXXXXX     ctr=XXXXXXXXXXXXXXXX      lr=XXXXXXXXXXXXXXXX

 cr=XXXXXXXX             cr0=?? cr1=?? cr2=?? cr3=?? cr4=?? cr5=?? cr6=?? cr7

 pc=XXXXXXXXXXXXXXXX

 Executable
```

Setting up your PC (4/5)

Now we want to close every tabs by going in Window > Close all. And exit every other tabs that stays open in prod debugger.

Now every tabs should be closed. Go in Window > New view.
Add a tab for:
- Memory (Change the columns to 16)
-Registers
-Breakpoints
-Dissasembly

Here is my layout:



<Untitled> - PS3 - ProDG Debugger for PS3 v420.1.15.3
File Debug Tools Window Help
Memory - 1 Byte Hex [PPU]
Address
Columns 16
Registers [PPC] [PPU]
Breakpoints [PPU]
Add
Disassembly [Disassembly] [PPU]
Stepping Mode : Dis

```
00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000200  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000210  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000220  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000230  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000240  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000250  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000260  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000270  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000280  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000290  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000002A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000002B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000002C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000002D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000002E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000002F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000300  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000310  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000320  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000330  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000340  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000350  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000360  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000370  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000380  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000390  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000003A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000003B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000003C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000003D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000003E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

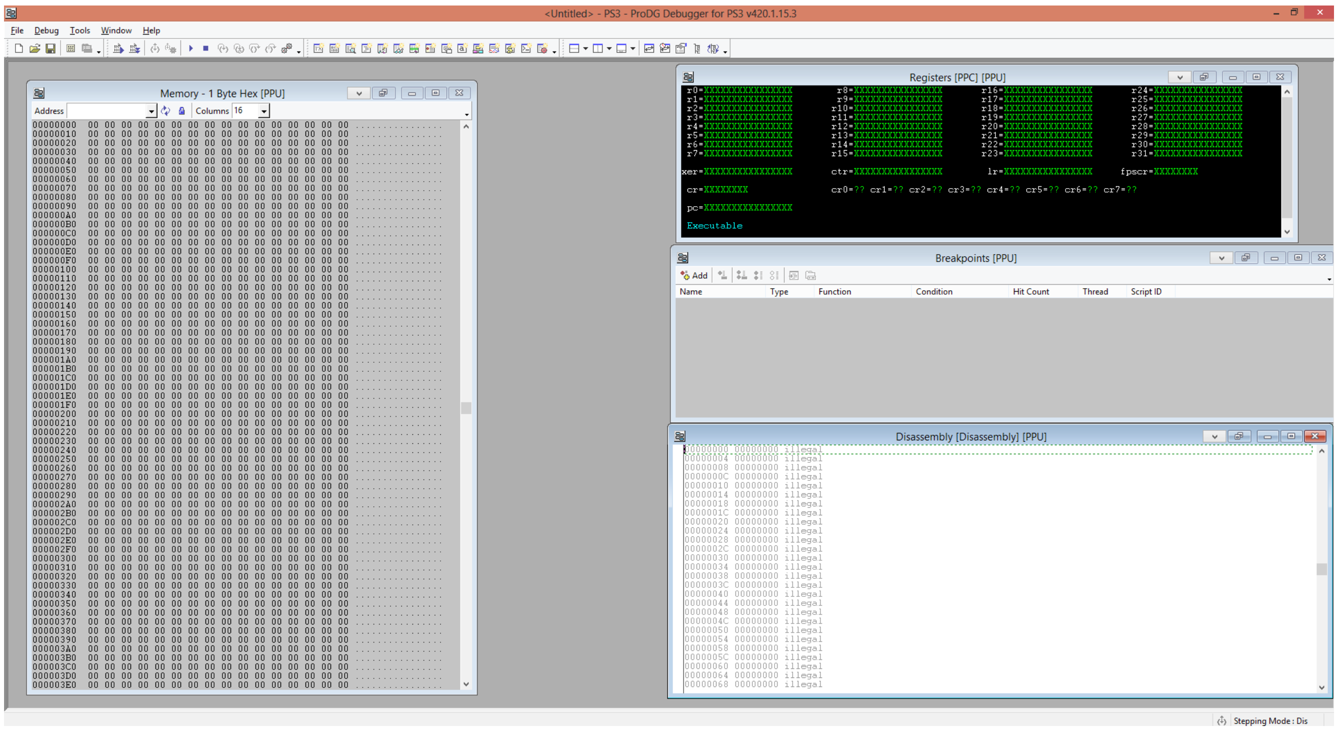
```
 r0=XXXXXXXXXXXXXXXX     r8=XXXXXXXXXXXXXXXX    r16=XXXXXXXXXXXXXXXX    r24=XXXXXXXXXXXXXXXX
 r1=XXXXXXXXXXXXXXXX     r9=XXXXXXXXXXXXXXXX    r17=XXXXXXXXXXXXXXXX    r25=XXXXXXXXXXXXXXXX
 r2=XXXXXXXXXXXXXXXX    r10=XXXXXXXXXXXXXXXX    r18=XXXXXXXXXXXXXXXX    r26=XXXXXXXXXXXXXXXX
 r3=XXXXXXXXXXXXXXXX    r11=XXXXXXXXXXXXXXXX    r19=XXXXXXXXXXXXXXXX    r27=XXXXXXXXXXXXXXXX
 r4=XXXXXXXXXXXXXXXX    r12=XXXXXXXXXXXXXXXX    r20=XXXXXXXXXXXXXXXX    r28=XXXXXXXXXXXXXXXX
 r5=XXXXXXXXXXXXXXXX    r13=XXXXXXXXXXXXXXXX    r21=XXXXXXXXXXXXXXXX    r29=XXXXXXXXXXXXXXXX
 r6=XXXXXXXXXXXXXXXX    r14=XXXXXXXXXXXXXXXX    r22=XXXXXXXXXXXXXXXX    r30=XXXXXXXXXXXXXXXX
 r7=XXXXXXXXXXXXXXXX    r15=XXXXXXXXXXXXXXXX    r23=XXXXXXXXXXXXXXXX    r31=XXXXXXXXXXXXXXXX

xer=XXXXXXXXXXXXXXXX    ctr=XXXXXXXXXXXXXXXX     lr=XXXXXXXXXXXXXXXX  fpscr=XXXXXXXX

 cr=XXXXXXXX            cr0=?? cr1=?? cr2=?? cr3=?? cr4=?? cr5=?? cr6=?? cr7=??

 pc=XXXXXXXXXXXXXXXX

 Executable
```

Name	Type	Function	Condition	Hit Count	Thread	Script ID

```
00000000 00000000 illegal
00000004 00000000 illegal
00000008 00000000 illegal
0000000C 00000000 illegal
00000010 00000000 illegal
00000014 00000000 illegal
00000018 00000000 illegal
0000001C 00000000 illegal
00000020 00000000 illegal
00000024 00000000 illegal
00000028 00000000 illegal
0000002C 00000000 illegal
00000030 00000000 illegal
00000034 00000000 illegal
00000038 00000000 illegal
0000003C 00000000 illegal
00000040 00000000 illegal
00000044 00000000 illegal
00000048 00000000 illegal
0000004C 00000000 illegal
00000050 00000000 illegal
00000054 00000000 illegal
00000058 00000000 illegal
0000005C 00000000 illegal
00000060 00000000 illegal
00000064 00000000 illegal
00000068 00000000 illegal
0000006C 00000000 illegal
```

Setting up your PC (5/5)

I also like to change the colors of my windows in Prodg Debugger to make it easier to read/organize. To change the colors of a window just go in tools > Options > Fonts and colors.


<Untitled> - PS3 - ProDG Debugger for PS3 v420.1.15.3
File
Debug
Tools
Window
Help
Memory - 1 Byte Hex [PPU]
Address
Columns 16
Registers [PPC] [PPU]
Breakpoints [PPU]
Add
Disassembly [Disassembly] [PPU]
Stepping Mode : Dis

```
00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000200  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000210  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000220  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000230  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000240  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000250  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000260  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000270  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000280  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000290  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000002A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000002B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000002C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000002D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000002E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000002F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000300  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000310  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000320  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000330  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000340  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000350  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000360  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000370  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000380  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000390  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000003A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000003B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000003C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000003D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000003E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
 r0=XXXXXXXXXXXXXXXX     r8=XXXXXXXXXXXXXXXX     r16=XXXXXXXXXXXXXXXX     r24=XXXXXXXXXXXXXXXX
 r1=XXXXXXXXXXXXXXXX     r9=XXXXXXXXXXXXXXXX     r17=XXXXXXXXXXXXXXXX     r25=XXXXXXXXXXXXXXXX
 r2=XXXXXXXXXXXXXXXX    r10=XXXXXXXXXXXXXXXX     r18=XXXXXXXXXXXXXXXX     r26=XXXXXXXXXXXXXXXX
 r3=XXXXXXXXXXXXXXXX    r11=XXXXXXXXXXXXXXXX     r19=XXXXXXXXXXXXXXXX     r27=XXXXXXXXXXXXXXXX
 r4=XXXXXXXXXXXXXXXX    r12=XXXXXXXXXXXXXXXX     r20=XXXXXXXXXXXXXXXX     r28=XXXXXXXXXXXXXXXX
 r5=XXXXXXXXXXXXXXXX    r13=XXXXXXXXXXXXXXXX     r21=XXXXXXXXXXXXXXXX     r29=XXXXXXXXXXXXXXXX
 r6=XXXXXXXXXXXXXXXX    r14=XXXXXXXXXXXXXXXX     r22=XXXXXXXXXXXXXXXX     r30=XXXXXXXXXXXXXXXX
 r7=XXXXXXXXXXXXXXXX    r15=XXXXXXXXXXXXXXXX     r23=XXXXXXXXXXXXXXXX     r31=XXXXXXXXXXXXXXXX

xer=XXXXXXXXXXXXXXXX    ctr=XXXXXXXXXXXXXXXX      lr=XXXXXXXXXXXXXXXX   fpscr=XXXXXXXX

 cr=XXXXXXXX            cr0=?? cr1=?? cr2=?? cr3=?? cr4=?? cr5=?? cr6=?? cr7=??

 pc=XXXXXXXXXXXXXXXX

 Executable
```

Name	Type	Function	Condition	Hit Count	Thread	Script ID

```
00000000 00000000 illegal
00000004 00000000 illegal
00000008 00000000 illegal
0000000C 00000000 illegal
00000010 00000000 illegal
00000014 00000000 illegal
00000018 00000000 illegal
0000001C 00000000 illegal
00000020 00000000 illegal
00000024 00000000 illegal
00000028 00000000 illegal
0000002C 00000000 illegal
00000030 00000000 illegal
00000034 00000000 illegal
00000038 00000000 illegal
0000003C 00000000 illegal
00000040 00000000 illegal
00000044 00000000 illegal
00000048 00000000 illegal
0000004C 00000000 illegal
00000050 00000000 illegal
00000054 00000000 illegal
00000058 00000000 illegal
0000005C 00000000 illegal
00000060 00000000 illegal
00000064 00000000 illegal
00000068 00000000 illegal
```

Now close Prodg Debugger to make it save the layout. That's it for the setting-up chapter of this tutorial.

Variables

Time to talk about the variables in PowerPC !
In C++ to declare a variable we do it like this:

```
int blb = 0x15;
```

In PowerPC we use registers. Those are the variables which can hold up to 8 bytes.
We will mainly use the last 4 bytes.
There is 32 registers that we can use. (not counting the link register and count register etc)

r0: Mostly used for the link register
r1: Stack pointer
r2: (read-only) We won't use it
r3: First argument of a function and also the 1st returned value from it !
r4: Second argument of a function and also the 2nd returned value from it !
r5-r10: Other parameters
r11-r31: General registers

Note:
r11: Used for syscalls
r13: its the same as r2

There is also the link register which is really important since it's a pointer that tells the program where to go after finishing a function.

When a function is called, a link register is generated. This register will point to the address that called the function + 0x4. Why +0x4 ? Because each instructions is 4 bytes and the program has to return to the next line else it will make an infinite loop which will call the same function over and over...

Example:

```
0x00: bl NullSub      //call NullSub and the link register will be 0x04 (next line)
0x04: li r3, 0        //this will be executed after the function returns.
```

Set a variable(1/2)

So now we need to know how to use these variables right ? Else it's pretty much useless.

So we will mainly use the register r3 To r10.
As I said we will also mainly focus on the last 4 bytes of the registers.

Try to imagine them like this:
XX XX YY YY

We will use `li` and `lis`.

`Li`: Load Immediate (YY YY)
`Lis`: Load Immediate Shifted (XX XX)

We can only set 2 bytes at once in PPC.

Examples:

```
                      XX XX YY YY
                      -- -- -- --
li r3, 0x15     //r3: 00 00 00 15
li r3, 0x1500   //r3: 00 00 15 00
lis r3, 0x15    //r3: 00 15 00 00
lis r3, 0x1500  //r3: 15 00 00 00

li r3, 0x1234   //r3: 00 00 12 34
lis r3, 0x1234  //r3: 12 34 00 00
```

Set a variable(2/2)

So now we can set 2 bytes in a register. What if we want to set a register up to 4 bytes? We will use `addic` and `addis`.

Addic will add a value to the YY YY part while Addis will add a value to the XX XX part.

Addic: Add immediate carrying
Addis: Add immediate Shifted

I will only use addic in this tutorial though but it was just so you know addis also exists which can turn out to be useful in some case.

Examples:

```
lis r3, 0x210        //r3: 02 10 00 00
addic r3, r3, 0x12   //r3: 02 10 00 12

li r3, 0x10          //r3: 00 00 00 10
addic r3, r3, 0x10   //r3: 00 00 00 20
```

Exercise:

I want r3 to be equal to 0xFCA280

Answer:

First we have to separate it in multiple bytes to make it easier. (start to separate from the right to the left)

0xFCA280 → FCA280 → FCA2 80 → FC A2 80 → 00 FC A2 80 (we add a 00 before it to reach the 4 bytes format)
Final Answer:

```
lis r3, 0xFC           //r3: 00 FC 00 00
addic r3, r3, 0xA280   //r3: 00 FC A2 80
```

Jumping

Now we can set variables like we want but what if we want to use jump in our codes?
We will use cmpwi, b, beq, bne, ble, bgt.

Cmpwi: Compare with immediate
b: Branch (no conditions, it jumps !)
beq: branch if equal
bne: Branch if not equal
ble: branch if less or equal
bgt: branch if greater of equal

C++ Version:

```
int BLBisHere = 1;

	if (BLBisHere == 1)
	{
		Welcome();
		 return;
	}
	else
	{
		return;
	}
```

PPC Version:

```
	li r3, 0x1     //BLBisHere
	cmpwi r3, 0x1  //compares r3 with 1
	 beq 0x08       //jumps 2 lines forward if equal (bl Welcome)
	 blr            //return
	 bl Welcome     //calls Welcome()
	 blr            //return
```

That's pretty much it with the conditions.

Reading in the memory(1/2)

Now we can do many things with PowerPC :) ! But what if we want to retrieve some data in the memory ? Well it's simple.

There is these instructions that we can use:

lbz | lhz | lwz | ld

Before explaining them let me show you the data formats on the PS3.

BYTE	= 1 byte	(BYTE)
HALF-WORD	= 2 bytes	(INT16)
WORD	= 4 bytes	(INT32)
DWORD	= 8 bytes	(INT64)

lbz: loads a byte
lhz: loads a half-word
lwz: loads a word
ld: loads a dword

Examples(for lbz, lwz and ld):

```
//lets say 0x2100000 contains: 12 34 56 78 00 00 00 00

lis r3, 0x210  //r3: 0x2100000

lbz r4, r3 //r4 = 00 00 00 12
lwz r5, r3 //r5: 12 34 56 78
 ld r6, r3 //r6: 12 34 56 78 00 00 00 00
```

Reading in the memory(2/2)

Now something really useful with there instructions is that we can use an a TEMPORARY addic in them. Why is it temporary? Because after the instruction read the memory, it will remove the addic that we used in the instruction.

Lets say we want to read the memory at: 0x12345678 (12 34 56 78)

```
lis r3, 0x1234 //r3 = 0x12340000
lwz r4, r3, 0x5678   //r4 is reading 4 bytes at: 0x12345678
lwz r5, r3 //r5 is reading 4 bytes at:0x12340000
(remember it was temporary)
```

This is a good way to save some lines and in modding when you're coding in PPC you WANT to save lines !!

Now, lets say we want to read a structure.

```
struct{
       int Age;  //4 bytes
       int Money; //4bytes
       int NumberOfHouses;
}UserInfos;
```

Lets say this structure is located at 0x2100000.

```
lis r3, 0x210         //r3: 0x2100000
lwz r4, r3        //r4: Age                 || 0x2100000
lwz r5, r3, 0x04  //r5: Money               || 0x2100004
lwz r6, r3, 0x08  //r6: NumberOfHouses      || 0x2100008
```

Writing in the memory

It is the same as reading in the memory !
We will use stb, sth, stw, std

stb: store byte
sth: store half-word
stw: store word
std: store dword

Examples:

We want to write the byte 0x01 at: 0x2100000

```
lis r3, 0x2100000
li r4, 0x01
stb r4, r3
```

We can also use the temporary addic with these.

Lets say r3 is the address to write, r4 the age, r5 the money and r6 the NumberOfHouses.

```
lis r3, 0x210   //r3: 0x2100000
li r4, 0x13     //r4: age  (0x13 is 19, remember those are HEX values)
li r5, 0x539    //r5: money (0x539 is 1337)
li r6, 0x2      //r6: NumberOfHouses(2 houses)

stw r4, r3      //store age at: 0x2100000
stw r5, r3, 0x4  //store age at: 0x2100004
stw r6, r3, 0x8  //store age at: 0x2100008
```

Creating our own Function

Now we can do a lot of things in PowerPC. But what about creating a function ?
Well it's simple, remember the link register in the 'variable' chapter ? It's time to use it.

```
stdu      r1, r1, -0x70          //allocate some space in the stack to store our stuff
mfspr     r0, LR           //get the link register and set r0 with it (r0 = LR)
std       r0, r1, 0x80    //store the link register in the stack

….......................     //Do everything you want here :)

ld        r0, r1, 0x80      //retrieve the link register
mtspr     LR, r0            //Set LR to the original link register
addi      r1, r1, 0x70      //Unallocate the stack
blr                         //Jump to the link register (return)
```

Now we have the base of our function. Lets make something simple.

C++:

```
int Add10(int a)
{
       return a + 0x10;
}
```

PPC:

```
stdu      r1, r1, -0x70          //allocate some space in the stack to store our stuff
mfspr     r0, LR           //get the link register and set r0 with it (r0 = LR)
std       r0, r1, 0x80    //store the link register in the stack

addic r3, r3, 0x10         //a + 0x10

ld        r0, r1, 0x80      //retrieve the link register
mtspr     LR, r0            //Set LR to the original link register
addi      r1, r1, 0x70      //Unallocate the stack
blr                         //Jump to the link register (return)
```

Of course in this case we don't need the stack and the link register stuff as we don't touch it or we don't call any other functions that could mess with it but this is how a function normally looks.

Remote Procedure Call(1/2)

Well with everything you've learned you can make a RPC for any games on PS3!
I will tell you about 2 instructions in PowerPC.

mtctr: Move to count register
bctr | bctrl: Branch to the count register || Call the count register

So lets say we will work at 0x2100000 and our RPC will take 3 args. (As I said it's a small RPC).

The RPC works like this:

```
-Read the 3 arguments
{
r3 at: 0x2100004
r4 at: 0x2100008
r5 at: 0x210000C
}
-Read the function to call
{
r6 at: 0x2100000
}
```

So the function will read the arguments and parse them in r3, r4 and r5. Then it will read the address to the function to call. (The function to call is stored as bytes at 0x2100000)

Remote Procedure Call(2/2)

Lets make the function :) !

```
stdu  r1, r1, -0x70          //allocate some space in the stack to store our stuff
mfspr     r0, LR            //get the link register and set r0 with it (r0 = LR)
std       r0, r1, 0x80    //store the link register in the stack

lis r6, 0x210              //Our memory spot
lwz r3, r6, 0x04           //r3: the value at 0x2100004
lwz r4, r6, 0x08           //r4: the value at 0x2100008
lwz r5, r6, 0x0C           //r5: the value at 0x210000C
lwz r6, r6                 //r6: the function address at: 0x2100000

mtctr r6                 //count register = the address at: 0x2100000
li r6, 0                   //we don't want r6 to ber parsed as an argument..
bctrl                    //Call the address stored in the count register
lis r6, 0x210
stw r3, r6, 0x10         //store the returned value at: 0x2100010
li r3, 0
stw r3, r6             //ERASE the function at:0x2100000 so it won't be called again !!

ld        r0, r1, 0x80      //retrieve the link register
mtspr     LR, r0            //Set LR to the original link register
addi      r1, r1, 0x70      //Unallocate the stack
blr                         //Jump to the link register (return)
```

Okay let's make make an example.

We have to call the function 'Welcome'. Welcome is located at: 0xFCA280.

It takes 2 arguments.

1st arg: int client
2nd arg: string WelcomeText

```
PS3::WriteInt32(0x2100004, 1);                 //client 1 at: 0x2100004
PS3::WriteString(0x2101000, "Hey BLB :D !");  //we write our string at: 0x2101000
PS3::WriteUInt32(0x2100008, 0x2101000);      //We write the address of the string at: 0x2100008
PS3::WriteUInt32(0x2100000, 0xFCA280);        //The address to call !!!
//there is nothing returned, so end of this mini RPC :D
```

Writing our PPC codes(1/7)

Now this is the long/hard part. We can write our stuff in PowerPC but we need to convert it to bytes now. You can either Google a PowerPC compiler or you can follow this tutorial.

Each instruction as an opcode, an opcode is the hex value of the instruction.

i will make a list of some opcodes, **to find any opcodes, just go in ida, click on an instruction and go to hex view.**

```
li = 0x38
lis = 0x3C
addic = 0x30
stb = 0x98
stw = 0x90
std = 0xF8
lbz = 0x88
lwz = 0x80
ld = 0xE8
cmpwi = 0x2C
b = 0x48 or 0x4B
bl = 0x48 or 0x4B
beq = 0x41, 0x82
bne = 0x40, 0x82
blt = 0x41, 0x80
bgt = 0x41, 0x81
mtctr = 0x7C, 0x69, 0x03, 0xA6
bctrl = 0x4E, 0x80, 0x04, 0x21
```

i will write the usage for all of them.

Writing our PPC codes(2/7)

Li/Lis

li

Code:

```
38 XX VV VV

38 = opcode
XX = Register to load the value into
VV VV = value to load in the register
```

Now i will explain the XX

you have to add 0x20 for each register

Code:

```
r0:  38 00 VV VV
r1:  38 20 VV VV
r2:  38 40 VV VV
r3:  38 60 VV VV
r4:  38 80 VV VV
r5:  38 A0 VV VV
r6:  38 C0 VV VV
r7:  38 E0 VV VV
```

Now for r8+ we need to add +1 to the opcode (0x38 + 0x1 = 0x39)

Code:

```
r8:  39 00 VV VV
r9:  39 20 VV VV
r10: 39 40 VV VV
r11: 39 60 VV VV
r12: 39 80 VV VV
```

i will stop at r12 , if you need to go higher just add +1 again

Writing our PPC codes(3/7)

lis:

lis is the same thing but with the opcode 3C, and 3D for r8+

Code:

```
r0:  3C 00 VV VV
r1:  3C 20 VV VV
r2:  3C 40 VV VV
r3:  3C 60 VV VV
r4:  3C 80 VV VV
r5:  3C A0 VV VV
r6:  3C C0 VV VV
r7:  3C E0 VV VV
```

Now for r8+ we need to add +1 to the opcode (0x38 + 0x1 = 0x39)

Code:

```
r8:  3D 00 VV VV
r9:  3D 20 VV VV
r10: 3D 40 VV VV
r11: 3D 60 VV VV
r12: 3D 80 VV VV
```

----

addic:

Code:

```
30 XY VV VV

38 = opcode
X = Register that will contain the result of the addition
Y = Register that were going to add to the value
VV VV = value to add to Y
```

Now for X, the register system is that same as li/lis
we add 0x20 and at r8 we change the opcode 30 to 31

but for Y, we just put the real number of the register

examples:

```
addic r3, r4, 0xFF ||   30 64 00 FF
```

```
addic r12, r4, 0xFF ||  31 84 00 FF
```

```
addic r3, r10, 0xFF || 30 6A 00 FF     ||  10 = 0x0A  (hexadecimal)
```

Writing our PPC codes(4/7)

stb,stw,std

Code:
```
stb = 0x98    // 0x99 for r8+
stw = 0x90    // 0x91 for r8+
std = 0xF8    // 0xF9 for r8+
```

i will use stw for the example.

They work like addis for the XY !

Code:
```
90 XY VV VV

X = register that will be sent in the memory   (VALUE)
Y = register of the address that will receive the VALUE (X)
VV VV = Temporary value to add to the address  (Y)
```

example:

Code:
```
lis r3, 0x2100000      || 3C 60 02 10
li r4, 0x15            || 38 80 00 15
stw r4, r3, 0x2101234  || 90 83 12 34
//ON THIS LINE, r3 RESETS BACK TO: 0x2100000 !!!
```

Code:
```
lbz = 0x88  // 0x89 for r8+
lwz = 0x80  // 0x81 for r8+
ld = 0xE8   // 0xE9 for r8+
```
usage (i will use lwz):

Code:
```
80 XY VV VV

It works like stw !!!

X = register that will contain the value read from the memory
Y = register of the address that will be read
VV VV = Temporary value to add to the address (Y)
```

Writing our PPC codes(5/7)

Example:

Code:

```
lis r3, 0x2100000      || 3C 60 02 10
li r4, 0x15            || 38 80 00 15
lwz r4, r3, 0x2101234  || 80 83 12 34
//ON THIS LINE, r3 RESETS BACK TO: 0x2100000  AND r4 = the first bytes that was at:
0x2101234 !!!
```

Code:

```
cmpwi = 0x2C

2c 0Y VV VV

0 = keep it as 0
Y = Register to compare, we just put its number in hex
VV VV = value that the register will be compared with
```

example:

Code:

```
cmpwi r3, 0x55  || 2c 03 00 55
```

other example:

Code:

```
cmwpi r12, 0x55 || 2c 0C 00 55    //0x0C = 12 in hexadecimal
```

Branching

Code:

```
b = 0x48 or 0x4B
bl = 0x48 or 0x4B
beq = 0x41, 0x82
bne = 0x40, 0x82
blt = 0x41, 0x80
bgt = 0x41, 0x81
```

Alright b and bl are tricky.
//current address = address where we are jumping from

Writing our PPC codes(6/7)

we use 48 when jumping to a location that is located AFTER the current address
we use 4B when jumping to a location that is located BEFORE the current address

Code:
```
48/4B XX XX XX

XX XX XX = difference between current position and the location we want to jump to


41 82 XX XX

XX XX = difference between current position and the location we want to jump to

also, to use bl we add +1 to the difference !!
```

example:

Code:
```
0x11010:  bl 0x11050   || 48 00 00 41
...
0x11050: //function...
```
// why 84 00 00 41 ? because 0x11050 - 0x11010 = 0x40 and to make it into a bl we need to add +1

0x40 + 1 = 0x41

so 48 00 00 41

and we use 48 because it is AFTER the current location (0x11010)

MTCTR/BCTRL

Now the last one, more complex

mtctr = 7C X9 03 A6

X = register to move to the count register, if r8+, 7C becomes 7D
we keep the rest as it is

Example

Code:
```
mtctr r4   || 7C 89 03 A6
mtctr r12  || 7D 89 03 A6
```

Writing our PPC codes(7/7)

bctrl = 4E 80 04 21
we keep it like this, BUT
bctrl is like bl, but we can also transform it
to bctr , which is like b

Code:

```
bctr    4E 80 04 20
bctrl   4E 80 04 21
```

End of lesson

So that was it for my PowerPC lessons. Try some stuff in the debugger and don't worry if you crash your ps3 20 times a day. It's normal :) We learn by our errors !

Also I would like to say a big thank you to:
aerosoul94
Choco
BadChoicesZ

Add me on Skype to check out my new clan who will help you with this lesson and more ! :)

Skype: BadLuckDobby