FROM EMOTION TO EMULATION
CELEBRATING 20 YEARS OF REVERSE ENGINEERING
- Core PCSX2 contributor
- Core PCSX2 contributor
- CS professor
- Core PCSX2 contributor
- CS professor
- VG industry contractor
# whoami

- Core PCSX2 contributor
- CS professor
- VG industry contractor
- Reverse engineer, console hacker
PCSX2
A Sony PlayStation 2 Emulator

• 98.24% of playable games!
PCSX2

A Sony PlayStation 2 Emulator

- 98.24% of playable games!
- Website's Alexa rank ~40k, most popular VG emulator I'm aware of
PCSX2
A Sony PlayStation 2 Emulator

- 98.24% of playable games!
- Website's Alexa rank ~40k, most popular VG emulator I'm aware of
- Very complex software project
PCSX2

A Sony PlayStation 2 Emulator

- 98.24% of playable games!
- Website's Alexa rank ~40k, most popular VG emulator I'm aware of
- Very complex software project
- 20 year old project
TECHNICALLY A PS2?
THAT’S STRETCHING IT!
AND A BLACK BOX TO EMULATE
How do we break into one?
How do we break into one?

We hack its web browser!
- fail0verflow, circa 2013
How do we break into one?
We hack its web browser!
- fail0verflow, circa 2013
How do we break into one?
We hack its web browser!
- fail0verflow, circa 2013
Us, circa 2002
Let's do it the good old way :)
BLACK BOX

Let's do it the good old way :)
Let's do it the good old way :)
Reversible logo!
WHAT IS THE EE
WHAT IS THE EE

More like what contains the EE
WHAT IS THE EE

More like what contains the EE

A core with 3 co-processors: (COP)
WHAT IS THE EE

More like what contains the EE

A core with 3 co-processors: (COP)

- COP0: System co-processor
**WHAT IS THE EE**

More like what contains the EE

A core with 3 co-processors: (COP)

- COP0: System co-processor
- COP1: A floating point unit (FPU)
WHAT IS THE EE

More like what contains the EE

A core with 3 co-processors: (COP)

- COP0: System co-processor
- COP1: A floating point unit (FPU)
- COP2: Vector Unit 0 Macro Mode (VU0 - Macro)
WHAT IS THE EE

More like what contains the EE

A core with 3 co-processors: (COP)

- COP0: System co-processor
- COP1: A floating point unit (FPU)
- COP2: Vector Unit 0 Macro Mode (VU0 - Macro)

Also partly designed by a chip designer called coolchips for his Master's thesis, pretty cool!
WHAT IS THE EE
WHAT IS THE EE

- Image Processing Unit (IPU)
WHAT IS THE EE

- Image Processing Unit (IPU)
- VPU1
WHAT IS THE EE

- Image Processing Unit (IPU)
- VPU1
- VPU0 with VU0 accessible as a COP
WHAT IS THE EE

- Image Processing Unit (IPU)
- VPU1
- VPU0 with VU0 accessible as a COP
- GIF/VIF (Graphics/Vector Interface)
WHAT IS THE EE

- Image Processing Unit (IPU)
- VPU1
- VPU0 with VU0 accessible as a COP
- GIF/VIF (Graphics/Vector Interface)
- i/dCache + ScratchPad (on die memory)
WHAT IS THE EE

- Image Processing Unit (IPU)
- VPU1
- VPU0 with VU0 accessible as a COP
- GIF/VIF (Graphics/Vector Interface)
- i/dCache + ScratchPad (on die memory)
- DMA Controller
WHAT IS THE EE

- Image Processing Unit (IPU)
- VPU1
- VPU0 with VU0 accessible as a COP
- GIF/VIF (Graphics/Vector Interface)
- i/dCache + ScratchPad (on die memory)
- DMA Controller

We're only getting started!
Only the best of MIPS have been used as we will see
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ADD</td>
<td>40. DSUBU</td>
<td>79. SRA</td>
</tr>
<tr>
<td>2. ADDI</td>
<td>41. JAL</td>
<td>80. SRAV</td>
</tr>
<tr>
<td>3. ADDIU</td>
<td>42. JALR</td>
<td>81. SRL</td>
</tr>
<tr>
<td>4. ADDU</td>
<td>43. JR</td>
<td>82. SRLV</td>
</tr>
<tr>
<td>5. AND</td>
<td>44. LB</td>
<td>83. SUB</td>
</tr>
<tr>
<td>6. ANDI</td>
<td>45. LBU</td>
<td>84. SUBU</td>
</tr>
<tr>
<td>7. BEQ</td>
<td>46. LD</td>
<td>85. SW</td>
</tr>
<tr>
<td>8. BEQL</td>
<td>47. LDL</td>
<td>86. SWL</td>
</tr>
<tr>
<td>9. BGEZ</td>
<td>48. LDR</td>
<td>87. SWR</td>
</tr>
<tr>
<td>10. BGEZAL</td>
<td>49. LH</td>
<td>88. SYNC.type</td>
</tr>
<tr>
<td>11. BGEZALL</td>
<td>50. LHU</td>
<td>89. SYS CALL</td>
</tr>
<tr>
<td>12. BGEZL</td>
<td>51. LUI</td>
<td>90. TEQ</td>
</tr>
<tr>
<td>13. BGTZ</td>
<td>52. LW</td>
<td>91. TEQI</td>
</tr>
<tr>
<td>14. BGTZL</td>
<td>53. LWL</td>
<td>92. TGE</td>
</tr>
<tr>
<td>15. BLEZ</td>
<td>54. LWL</td>
<td>93. TGEI</td>
</tr>
<tr>
<td>16. BLEZL</td>
<td>55. LWU</td>
<td>94. TGEIU</td>
</tr>
<tr>
<td>17. BLTZ</td>
<td>56. MFIH</td>
<td>95. TGEU</td>
</tr>
<tr>
<td>18. BLTZAL</td>
<td>57. MFLO</td>
<td>96. TL</td>
</tr>
<tr>
<td>19. BLTZALL</td>
<td>58. MOVN</td>
<td>97. TLTI</td>
</tr>
<tr>
<td>20. BLTZL</td>
<td>59. MOVT</td>
<td>98. TLTIU</td>
</tr>
<tr>
<td>21. BNE</td>
<td>60. MTHI</td>
<td>99. TLTU</td>
</tr>
<tr>
<td>22. BNE.L</td>
<td>61. MTHL</td>
<td>100. TNE</td>
</tr>
<tr>
<td>23. BREAK</td>
<td>62. MULT</td>
<td>101. TNEI</td>
</tr>
<tr>
<td>24. DADD</td>
<td>63. MULTU</td>
<td>102. XOR</td>
</tr>
<tr>
<td>25. DADDI</td>
<td>64. NOR</td>
<td>103. XORI</td>
</tr>
<tr>
<td>26. DADDIU</td>
<td>65. OR</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>Add</td>
<td></td>
</tr>
<tr>
<td>ANI</td>
<td>AND</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>AND</td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>Subtract</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>XOR</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>NOT</td>
<td></td>
</tr>
<tr>
<td>EQ</td>
<td>Equal</td>
<td></td>
</tr>
<tr>
<td>NE</td>
<td>Not equal</td>
<td></td>
</tr>
<tr>
<td>GE</td>
<td>Greater than or equal</td>
<td></td>
</tr>
<tr>
<td>LE</td>
<td>Less than or equal</td>
<td></td>
</tr>
<tr>
<td>DIV1</td>
<td>Divide 1</td>
<td></td>
</tr>
<tr>
<td>DIVU1</td>
<td>Divide unsigned 1</td>
<td></td>
</tr>
<tr>
<td>LO</td>
<td>Less than or equal</td>
<td></td>
</tr>
<tr>
<td>MADD</td>
<td>Multiply and add</td>
<td></td>
</tr>
<tr>
<td>MADD1</td>
<td>Multiply and add 1</td>
<td></td>
</tr>
<tr>
<td>MADDU</td>
<td>Multiply and add unsigned</td>
<td></td>
</tr>
<tr>
<td>PECM</td>
<td>Subtract with borrow</td>
<td></td>
</tr>
<tr>
<td>MFH</td>
<td>Move forward high</td>
<td></td>
</tr>
<tr>
<td>MLO</td>
<td>Move low</td>
<td></td>
</tr>
<tr>
<td>MSEW</td>
<td>Move sign extend wider</td>
<td></td>
</tr>
<tr>
<td>MF</td>
<td>Move forward</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Move</td>
<td></td>
</tr>
<tr>
<td>MTS</td>
<td>Multiply and subtract</td>
<td></td>
</tr>
<tr>
<td>MTSB</td>
<td>Multiply and subtract borrow</td>
<td></td>
</tr>
<tr>
<td>MTSNH</td>
<td>Multiply and subtract high</td>
<td></td>
</tr>
<tr>
<td>MTSNL</td>
<td>Multiply and subtract low</td>
<td></td>
</tr>
<tr>
<td>MTL</td>
<td>Multiply and subtract low</td>
<td></td>
</tr>
<tr>
<td>MTSU</td>
<td>Multiply and subtract high</td>
<td></td>
</tr>
<tr>
<td>MULTI</td>
<td>Multiply</td>
<td></td>
</tr>
<tr>
<td>MULT1</td>
<td>Multiply 1</td>
<td></td>
</tr>
<tr>
<td>MULTU</td>
<td>Multiply unsigned</td>
<td></td>
</tr>
<tr>
<td>MULTU1</td>
<td>Multiply unsigned 1</td>
<td></td>
</tr>
<tr>
<td>PB</td>
<td>Push</td>
<td></td>
</tr>
<tr>
<td>PBW</td>
<td>Push with borrow</td>
<td></td>
</tr>
<tr>
<td>PADD</td>
<td>Pre-decrement and add</td>
<td></td>
</tr>
<tr>
<td>PADDH</td>
<td>Pre-decrement and add high</td>
<td></td>
</tr>
<tr>
<td>PADDW</td>
<td>Pre-decrement and add low</td>
<td></td>
</tr>
<tr>
<td>PADDU</td>
<td>Pre-decrement and add unsigned</td>
<td></td>
</tr>
<tr>
<td>PADDUH</td>
<td>Pre-decrement and add unsigned high</td>
<td></td>
</tr>
<tr>
<td>PADDUW</td>
<td>Pre-decrement and add unsigned low</td>
<td></td>
</tr>
<tr>
<td>PADW</td>
<td>Pre-decrement and add</td>
<td></td>
</tr>
<tr>
<td>PADSB</td>
<td>Pre-decrement and subtract with borrow</td>
<td></td>
</tr>
<tr>
<td>PADSH</td>
<td>Pre-decrement and subtract with high</td>
<td></td>
</tr>
<tr>
<td>PAND</td>
<td>Pre-decrement and and</td>
<td></td>
</tr>
<tr>
<td>PCEQB</td>
<td>Pre-decrement and exchange byte high</td>
<td></td>
</tr>
<tr>
<td>PCEQH</td>
<td>Pre-decrement and exchange high</td>
<td></td>
</tr>
<tr>
<td>PCQW</td>
<td>Pre-decrement and clear word</td>
<td></td>
</tr>
<tr>
<td>PCTB</td>
<td>Pre-decrement and test byte</td>
<td></td>
</tr>
<tr>
<td>PCGT</td>
<td>Pre-decrement and test</td>
<td></td>
</tr>
<tr>
<td>PCGTW</td>
<td>Pre-decrement and test word</td>
<td></td>
</tr>
<tr>
<td>PCPYH</td>
<td>Pre-decrement and push high</td>
<td></td>
</tr>
<tr>
<td>PMTL0</td>
<td>Push memory low 0</td>
<td></td>
</tr>
<tr>
<td>PMULT</td>
<td>Push memory low 1</td>
<td></td>
</tr>
<tr>
<td>PMULTUw</td>
<td>Push memory low unsigned 1</td>
<td></td>
</tr>
<tr>
<td>PMULTW</td>
<td>Push memory low unsigned</td>
<td></td>
</tr>
<tr>
<td>PNOR</td>
<td>Push not or</td>
<td></td>
</tr>
<tr>
<td>POR</td>
<td>Push or or</td>
<td></td>
</tr>
<tr>
<td>PPAC5</td>
<td>Push processor access 5</td>
<td></td>
</tr>
<tr>
<td>PPACB</td>
<td>Push processor access 6</td>
<td></td>
</tr>
<tr>
<td>PPA</td>
<td>Push processor access</td>
<td></td>
</tr>
<tr>
<td>PPACW</td>
<td>Push processor access word</td>
<td></td>
</tr>
<tr>
<td>PREV</td>
<td>Push previous</td>
<td></td>
</tr>
<tr>
<td>PROT3W</td>
<td>Push protected 3 word</td>
<td></td>
</tr>
<tr>
<td>PSL</td>
<td>Push save</td>
<td></td>
</tr>
<tr>
<td>PSLV</td>
<td>Push save low</td>
<td></td>
</tr>
<tr>
<td>PSLW</td>
<td>Push save low word</td>
<td></td>
</tr>
<tr>
<td>PSSRAH</td>
<td>Push save stack address high</td>
<td></td>
</tr>
<tr>
<td>PSSRAW</td>
<td>Push save stack address word</td>
<td></td>
</tr>
<tr>
<td>PSRAW</td>
<td>Push save stack address</td>
<td></td>
</tr>
<tr>
<td>PSL</td>
<td>Push stack</td>
<td></td>
</tr>
<tr>
<td>PSLV</td>
<td>Push stack low</td>
<td></td>
</tr>
<tr>
<td>PSLW</td>
<td>Push stack low word</td>
<td></td>
</tr>
<tr>
<td>PSUBB</td>
<td>Push subbase</td>
<td></td>
</tr>
<tr>
<td>PSUB</td>
<td>Push sub</td>
<td></td>
</tr>
<tr>
<td>PSUBS</td>
<td>Push subbase signed</td>
<td></td>
</tr>
<tr>
<td>PSUBSH</td>
<td>Push subbase high</td>
<td></td>
</tr>
<tr>
<td>PSUBSW</td>
<td>Push subbase word</td>
<td></td>
</tr>
<tr>
<td>PSUBU</td>
<td>Push sub</td>
<td></td>
</tr>
<tr>
<td>PSUBUW</td>
<td>Push sub unsigned word</td>
<td></td>
</tr>
<tr>
<td>PSUBW</td>
<td>Push sub unsigned</td>
<td></td>
</tr>
<tr>
<td>PXOR</td>
<td>Push xor</td>
<td></td>
</tr>
<tr>
<td>QFSRV</td>
<td>Queue full service request valid</td>
<td></td>
</tr>
<tr>
<td>SQ</td>
<td>Stop</td>
<td></td>
</tr>
</tbody>
</table>

**EE Core additions**
This is the MIPS part!
1 lw t5,0x0(t7) ; t5 = MEM[t7]
2 jr t5 ; jump to t5
3 addiu t5,t5,4 ; t5+=4
1 lw t5,0x0(t7) ; t5 = MEM[t7]
2 jr t5 ; jump to t5
3 addiu t5,t5,4 ; t5+=4
1  lw  t5,0x0(t7) ; t5 = MEM[t7]
2  jr  t5 ; jump to t5
3  addiu t5,t5,4 ; t5+=4
1 lw t5,0x0(t7) ; t5 = MEM[t7]
2 jr t5 ; jump to t5
3 addiu t5,t5,4 ; t5+=4
A CPU executes instructions by passing through multiple steps.
A CPU executes instructions by passing through multiple steps

We call those a pipeline
A CPU executes instructions by passing through multiple steps.

We call those a pipeline.
A CPU executes instructions by passing through multiple steps

We call those a pipeline
A CPU executes instructions by passing through multiple steps.

We call those a pipeline.
A CPU executes instructions by passing through multiple steps.

We call those a pipeline.
A CPU executes instructions by passing through multiple steps.

We call those a pipeline.

Not exactly true but it'll work for now.
A CPU executes instructions by passing through multiple steps

We call those a pipeline

Not exactly true but it'll work for now

The execute step will also be used as memory access
Here's an example

78 00 b3 ff

s3 = 00
Here's an example

```
sd s3, 0x78(sp)
```
Here's an example

sd s3, FF
Here's an example

sd s3,
Here's an example

In reality all of those steps are executed in parallel on multiple instructions
Here's an example

IR = Instruction Register, current instruction
Here's an example

IR = Instruction Register, current instruction

What if this is a jump?
Here's an example

IR = Instruction Register, current instruction

What if this is a jump?

s3 = FF
Here's an example:

Instead of wasting 2 steps, MIPS decided to execute an instruction out of order to waste 1.

IR = Instruction Register, current instruction

What if this is a jump?
Here's an example

Instead of wasting 2 steps, MIPS decided to execute an instruction out of order to waste 1

IR = Instruction Register, current instruction

What if this is a jump?
THE DELAY SLOTS STRIKES BACK

1 lui $a0, 0
2 li $v1, FlushCache
3 syscall
4 li $v1, ResetEE
THE DELAY SLOTS STRIKES BACK

1. lui $a0, 0
2. li $v1, FlushCache
3. syscall
4. li $v1, ResetEE
Pipeline

The Delay Slots Strikes Back

```
1 lui $a0, 0
2 li $v1, FlushCache
3 syscall
4 li $v1, ResetEE
```
THE DELAY SLOTS STRIKES BACK

1 `lui $a0, 0`
2 `li $v1, FlushCache`
3 `syscall`
4 `li $v1, ResetEE`
THE DELAY SLOTS STRIKES BACK

PIPELINE

1 lui $a0, 0
2 li $v1, FlushCache
3 syscall
4 li $v1, ResetEE
INTERRUPTS

PIPELINE
INTERRUPTS

syscall is like an interrupt instruction
INTERRUPTS

syscall is like an interrupt instruction

The CPU switches to kernel mode and drops the entire pipeline
INTERRUPTS

syscall is like an interrupt instruction

The CPU switches to kernel mode and drops the entire pipeline

Everything gets fetched back again after the syscall is done
COP

COPO
Handles multiple system things:
COPO

Handles multiple system things:

- Memory Management
COPO

Handles multiple system things:

- Memory Management
- Exceptions
Handles multiple system things:

- Memory Management
- Exceptions
- Debugging
Handles multiple system things:

- Memory Management
- Exceptions
- Debugging
- Cache
COPO

Handles multiple system things:

- Memory Management
- Exceptions
- Debugging
- Cache
- Interrupts! (Nice transition)
COP

COP1
A Floating Point Unit (FPU)
A Floating Point Unit (FPU)
(this thing)

0.3921230137348175048828125
0x3ec8c459
A Floating Point Unit (FPU) (this thing)

COP1

0.3921230137348175048828125
0x3ec8c459

Not IEEE 754 compliant!!
A Floating Point Unit (FPU) (this thing)

COP1

0.3921230137348175048828125
0x3ec8c459

Not IEEE 754 compliant!!

Relevant list of features not implemented:
A Floating Point Unit (FPU) (this thing)

| 0.3921230137348175048828125 | 0x3ec8c459 |

Not IEEE 754 compliant!!

Relevant list of features not implemented:

- NaN
A Floating Point Unit (FPU)
(this thing)

0.3921230137348175048828125
0x3ec8c459

Not IEEE 754 compliant!!

Relevant list of features not implemented:

- NaN
- Nearest roundings
A Floating Point Unit (FPU) (this thing)

<table>
<thead>
<tr>
<th>0.3921230137348175048828125</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3ec8c459</td>
</tr>
</tbody>
</table>

Not IEEE 754 compliant!!

Relevant list of features not implemented:

- NaN
- Nearest roundings
- +/- ∞
A Floating Point Unit (FPU) (this thing)

Not IEEE 754 compliant!!

Relevant list of features not implemented:

- NaN
- Nearest roundings
- +/- \( \infty \)
- Exceptions

0.3921230137348175048828125
0x3ec8c459
A Floating Point Unit (FPU) (this thing)

\[
\begin{array}{ll}
0.3921230137348175048828125 & 0x3ec8c459 \\
\end{array}
\]

Not IEEE 754 compliant!!

Relevant list of features not implemented:

- NaN
- Nearest roundings
- +/- \( \infty \)
- Exceptions
- Denormalized numbers
A Floating Point Unit (FPU) (this thing)

0.3921230137348175048828125
0x3ec8c459

Not IEEE 754 compliant!!

Relevant list of features not implemented:

- NaN
- Nearest roundings
- +/- ∞
- Exceptions
- Denormalized numbers

Result: an absolute pain in the ass to emulate
Two of them, composed of two things:

- A Vector Unit (VU)
Two of them, composed of two things:

- A Vector Unit (VU)
- A Vector Interface (VIF)
Two of them, composed of two things:

- A Vector Unit (VU)
- A Vector Interface (VIF)

VPU0 can either work as a COP or as a microprocessor
Two of them, composed of two things:

- A Vector Unit (VU)
- A Vector Interface (VIF)

VPU0 can either work as a COP or as a microprocessor.

If it runs in COP(macro) mode, it will act as a superset of instructions for the EE core.
Two of them, composed of two things:

- A Vector Unit (VU)
- A Vector Interface (VIF)

VPU0 can either work as a COP or as a microprocessor.

If it runs in COP(macro) mode, it will act as a superset of instructions for the EE core.

Otherwise it will execute instructions in parallel fed in a microprogram by the EE.
VPU1 can transfer directly to the GS memory by using 2 methods:
VPU1 can transfer directly to the GS memory by using 2 methods:

- XGKICK (Path 1)
VPU1 can transfer directly to the GS memory by using 2 methods:

- XGKICK (Path 1)
- VIF1 (Path 2)
VPU1 can transfer directly to the GS memory by using 2 methods:

- XGKICK (Path 1)
- VIF1 (Path 2)

The EE and the VU1 uses a third method to transfer data to the GPU, the GIF.
VPU1 can transfer directly to the GS memory by using 2 methods:

- XGKICK (Path 1)
- VIF1 (Path 2)

The EE and the VU1 use a third method to transfer data to the GPU, the GIF

NB: Path 1 and Path 2 also use the GIF but have higher priority, confusing yet?
VPU - EXAMPLE
VPU

The EE sends the model data to the VIF
The EE sends the model data to the VIF

```
.align 0
;test.dae_mpi_pkt1.obj
;Automatically generated by kh2vif
;kh2vif by GovanifY ~ 2017
stcycl 01, 01

unpack[r] V4_32, 0, *
.int 1, 0, 0, 0
.int 36, 4, 54, 56
.int 0, 0, 0, 0
.int 14, 40, 0, 5
.EndUnpack

stcycl 01, 01

unpack[r] V2_16, 4, *
.short 2048, 0
.short 1024, 1024
.short 1024, 0
.short 1024, 3071
.short 2048, 2048
.short 2048, 3071
.short 3071, 2048
```
The EE sends the model data to the VIF

```
1 .align 0
2 ;test.dae_mp1_pkt1.obj
3 ;Automatically generated by kh2vif
4 ;kh2vif by GovanifY ~ 2017
5 stcyc1 01, 01
6
7 unpack[r] V4_32, 0, *
8 .int 1, 0, 0, 0
9 .int 36, 4, 54, 56
10 .int 0, 0, 0, 0
11 .int 14, 40, 0, 5
12 .EndUnpack
13
14 stcyc1 01, 01
15
16 unpack[r] V2_16, 4, *
17 .short 2048, 0
18 .short 1024, 1024
19 .short 1024, 0
20 .short 1024, 3071
21 .short 2048, 2048
22 .short 2048, 3071
23 .short 3071, 2048
```
The EE sends the model data to the VIF

```
1 .align 0
2 ;test.dae_mp1_pkt1.obj
3 ;Automatically generated by kh2vif
4 ;kh2vif by GovanifY ~ 2017
5 stcyc1 01, 01
6
7 unpack[r] V4_32, 0, *
8 .int 1, 0, 0, 0
9 .int 36, 4, 54, 56
10 .int 0, 0, 0, 0
11 .int 14, 40, 0, 5
12 .EndUnpack
13
14 stcyc1 01, 01
15
16 unpack[r] V2_16, 4, *
17 .short 2048, 0
18 .short 1024, 1024
19 .short 1024, 0
20 .short 1024, 3071
21 .short 2048, 2048
22 .short 2048, 3071
23 .short 3071, 2048
```
The EE sends the model data to the VIF

```assembly
.align 0
@test.dae_mp1_pkt1.obj
;Automatically generated by kh2vif
;kh2vif by GovanifY ~ 2017
stcycl 01, 01
.unpack[r] V4_32, 0, *
.int 1, 0, 0, 0
.int 36, 4, 54, 56
.int 0, 0, 0, 0
.int 14, 40, 0, 5
.EndUnpack
stcycl 01, 01
.unpack[r] V2_16, 4, *
.short 2048, 0
.short 1024, 1024
.short 1024, 0
.short 1024, 3071
.short 2048, 2048
.short 2048, 3071
.short 3071, 2048
```

The VIF1 executes the unpack commands and writes the data to its memory

The VIF1 sends the final data to the GS for rasterization

The VU1 transforms the data and calculates relative positions
SPU

SPU2
Based on the PS1 SPU, but with 2 cores!
Based on the PS1 SPU, but with 2 cores!
Based on the PS1 SPU, but with 2 cores!

Has customizable IRQ!!
Based on the PS1 SPU, but with 2 cores!

Has customizable IRQ!!

Games use them as highly precise interrupts by setting an IRQ at a write-back address used during the mixing stage.
Based on the PS1 SPU, but with 2 cores!

Has customizable IRQ!!

Games use them as highly precise interrupts by setting an IRQ at a write-back address used during the mixing stage.

The mixer has a sample rate of 48kHZ in PS2 mode, 44.1 in PS1 compatible mode.
Also has a Schroeder Reverberator!

Uses 4 parallel comb filters in a rotating buffer
Also has a Schroeder Reverberator!

Uses 4 parallel comb filters in a rotating buffer

 Adds gain, then mixes back with the original input, rewriting the rotating buffer in the process
Also has a Schroeder Reverberator!

Uses 4 parallel comb filters in a rotating buffer.

Adds gain, then mixes back with the original input, rewriting the rotating buffer in the process.
WHAT IS THE IOP
Good question! It's a MIPS-based processor...
WHAT IS THE IOP

Good question! It's a MIPS-based processor...
... or PowerPC.
WHAT IS THE IOP

Good question! It's a MIPS-based processor...
... or PowerPC.

wait...WHAT?
WHAT IS THE IOP

Good question! It's a MIPS-based processor...
... or PowerPC.

wait...WHAT?

... Let's ignore that for now.
WHAT IS THE IOP

... Let's ignore that for now.
WHAT IS THE IOP

Good question! It's a MIPS-based processor...
... Let's ignore that for now.

It is the PS1 CPU, just repurposed in order to handle all the I/O, devices and drivers in the PS2.
WHAT IS THE IOP

Good question! It's a MIPS-based processor...
... Let's ignore that for now.

It is the PS1 CPU, just repurposed in order to handle all the I/O, devices and drivers in the PS2.

The EE and the IOP communicate through the Subsystem Interface (SIF).
WHAT IS THE IOP
WHAT IS THE IOP

A MIPS I "compatible" CPUs with 2 COP
WHAT IS THE IOP

A MIPS I "compatible" CPUs with 2 COP

• COP0: System Management
WHAT IS THE IOP

A MIPS I "compatible" CPUs with 2 COP

- COP0: System Management
- COP2: Geometry Transformation Engine (GTE)
WHAT IS THE IOP

A MIPS I "compatible" CPUs with 2 COP

• COP0: System Management
• COP1: ???
• COP2: Geometry Transformation Engine (GTE)
WHAT IS THE IOP

A MIPS I "compatible" CPUs with 2 COP

- COP0: System Management
- COP1: ???
- COP2: Geometry Transformation Engine (GTE)

Sony doesn't know how to count
YOU CAN'T HAVE AN INSANE FPU
IF YOU DON'T HAVE ONE
Dynamic Memory Allocation and the Heap

Memory allocation functions allow the programmer to increase the depth and variety of the game world whilst making the best use of the PlayStation’s relatively modest memory size.

In C, memory allocation is achieved using the `malloc` and `free` functions. Unfortunately there have been several problems with these functions on PlayStation.

The standard `malloc/free` combination supplied as part of `libc` fragments memory due to a bug in the `free` function. This means that large chunks on the machine memory become inaccessible even though they are not holding any valid data.
ONE LAST THING

Dynamic Memory Allocation and the Heap

Memory allocation functions allow the programmer to increase the depth and variety of the game world whilst making the best use of the PlayStation’s relatively modest memory size.

In C, memory allocation is achieved using the `malloc` and `free` functions. Unfortunately there have been several problems with these functions on PlayStation.

The standard `malloc/free` combination supplied as part of `libc` fragments memory due to a bug in the `free` function. This means that large chunks on the machine memory become inaccessible even though they are not holding any valid data.

PS1: Bring out your own f---ing chip and system libraries
Meet PS2 Slim hardware!
Slim, right?

Meet PS2 Slim hardware!
DECKARD

Slim, right?

Wanna know why?

Meet PS2 Slim hardware!
DECKARD
The combined EE+GS+RDRAM+DRAM found in the SCPH-7900x and SCPH-9000x series
A PS2 ON-A-CHIP IS NOT ENOUGH
A PS2 ON-A-CHIP IS NOT ENOUGH

Meet deckard!
A PS2 ON-A-CHIP IS NOT ENOUGH

Meet deckard!

A PowerPC based replacement for the IOP
A PS2 ON-A-CHIP IS NOT ENOUGH

Meet deckard!

A PowerPC based replacement for the IOP

Emulates PS1 features through software
A PS2 ON-A-CHIP IS NOT ENOUGH

Meet deckard!

A PowerPC based replacement for the IOP

Emulates PS1 features through software

Fortunately we don't care about it, we are writing an emulator, not trying to emulate the emulator emulating the console :D
DRM

COPY PROTECTION
About time we talk about it
COPY PROTECTION

About time we talk about it

Essentially a mod of PS1's copy protection
About time we talk about it
Essentially a mod of PS1's copy protection

The PS1 replaced CD's ATIP (which is a sinusoidal constant of ~22kHZ) by their own region specific constant
About time we talk about it

Essentially a mod of PS1's copy protection

The PS1 replaced CD's ATIP (which is a sinusoidal constant of ~22kHZ) by their own region specific constant

The ATIP is normally used by players to synchronize their laser's timing
About time we talk about it

Essentially a mod of PS1's copy protection

The PS1 replaced CD's ATIP (which is a sinusoidal constant of ~22kHZ) by their own region specific constant

The ATIP is normally used by players to synchronize their laser's timing

Data can also be stored by modulating the ATIP +/- 1kHZ!
The ATIP is around there
DRM

COPY PROTECTION
The PS2 instead stores the Title ID of the disc in it.
COPY PROTECTION

The PS2 instead stores the Title ID of the disc in it.

Mechacon then derives an encryption key out of the Title ID which will be used to decrypt and verify the disc.
The PS2 instead stores the Title ID of the disc in it.

Mechacon then derives an encryption key out of the Title ID which will be used to decrypt and verify the disc.

It will then proceed to decrypt the "PlayStation 2" logo you see at each boot once sent to it.
The PS2 instead stores the Title ID of the disc in it.

Mechacon then derives an encryption key out of the Title ID which will be used to decrypt and verify the disc.

It will then proceed to decrypt the "PlayStation 2" logo you see at each boot once sent to it.

...But we can completely ignore this by skipping the verification logic in the BIOS!
DRM

COPY PROTECTION
Sony tries to make this harder by making it harder to power on the mechacon.
COPY PROTECTION

Sony tries to make this harder by making it harder to power on the mechacon

...forgetting you could just dump the BIOS out of your flash chip and reverse engineer it
Sony tries to make this harder by making it harder to power on the mechacon

...forgetting you could just dump the BIOS out of your flash chip and reverse engineer it

......and that the bootloader verifies the integrity of your BIOS with a simple CRC which is prone to collisions
Sony tries to make this harder by making it harder to power on the mechacon

...forgetting you could just dump the BIOS out of your flash chip and reverse engineer it

......and that the bootloader verifies the integrity of your BIOS with a simple CRC which is prone to collisions

The mechacon is essentially a security processor that you can completely ignore and useless in functionality
```c
void mechaconAuth()
{
    int k;

    while (cdvdRead(0x17) != 0x40) {;}

    cdvdWrite(0x17, 0);

    cdvdWrite(0x16, 0x80);

    while (cdvdRead(0x16) != 0x80) {;}
    while (cdvdRead(0x17) != 0x40)
    {
        cdvdRead(0x18);
    }

    while (cdvdRead(0x17) != 0x40) {;}
    cdvdWrite(0x16, 0x81);
    while (cdvdRead(0x16) != 0x81) {;}
    while (cdvdRead(0x17) != 0x40)
    {
        cdvdRead(0x18);
    }

    while (cdvdRead(0x17) != 0x40) {;}

    for (k = 0; k < 16; k++)
    {
        cdvdWrite(0x17, 0xff);
    }
}
```
Good thing we can just ignore it when emulating!
Good thing we can just ignore it when emulating!

```c
1 case 0x80:
  // secrman: _mechacon_auth_0x80
2     SetResultSize(1);  //in:1
3     cdvd.mg_datatype = 0;  //data
4     cdvd.Result[0] = 0;
5     break;
6
7 case 0x81:
  // secrman: _mechacon_auth_0x81
8     SetResultSize(1);  //in:1
9     cdvd.mg_datatype = 0;  //data
10    cdvd.Result[0] = 0;
11    break;
12
13 case 0x82:
  // secrman: _mechacon_auth_0x82
14     SetResultSize(1);  //in:16
15    cdvd.Result[0] = 0;
16    break;
```
Good thing we can just ignore it when emulating!

```
1  case 0x80: // secrman: __mechacon_auth_0x80
2    SetResultSize(1); //in:1
3    cdvd.mg_datatype = 0; //data
4    cdvd.Result[0] = 0;
5    break;
6
7  case 0x81: // secrman: __mechacon_auth_0x81
8    SetResultSize(1); //in:1
9    cdvd.mg_datatype = 0; //data
10   cdvd.Result[0] = 0;
11   break;
12
13  case 0x82: // secrman: __mechacon_auth_0x82
14    SetResultSize(1); //in:16
15    cdvd.Result[0] = 0;
16    break;
```

Just have to make sure to return the nice values for the BIOS
DRM

COPY PROTECTION
What would be Sony's copy protection without trademark infringement?
What would be Sony's copy protection without trademark infringement?
What would be Sony's copy protection without trademark infringement?
What would be Sony's copy protection without trademark infringement?

```
// letters = SLES, numbers = 54232
key[0] = ((numbers & 0x1F) << 3) | ((0xFFFFFFFF & letters) >> 25);
key[1] = (numbers >> 10) | ((0xFFFFFFFF & letters) << 7);
key[2] = ((numbers & 0x3E0) >> 2) | 0x04;
```
What would be Sony's copy protection without trademark infringement?

```
// letters = SLES, numbers = 54232
key[0] = ((numbers & 0x1F) << 3) | ((0xFFFFFFFF & letters) >> 25);
key[1] = (numbers >> 10) | ((0xFFFFFFFF & letters) << 7);
key[2] = ((numbers & 0x3E0) >> 2) | 0x04;
```

```
for(int i=0; i<12*2048; i++)
{
    logo[i] = ((logo[i]<<5)|(logo[i]>>3)) ^ magicNum;
}
```
What would be Sony's copy protection without trademark infringement?

Also differs between regions
DRM

COPY PROTECTION

11101010
for (int i = 0; i < 12 * 2048; i++)
{
    logo[i] = ((logo[i] << 5) | (logo[i] >> 3)) ^ magicNum;
}
for(int i=0; i<12*2048; i++)
{
    logo[i] = ((logo[i]<<5)|(logo[i]>>3)) ^ magicNum;
}
DRM

COPY PROTECTION

```c
for(int i=0; i<12*2048; i++)
{
    logo[i] = ((logo[i]<<5)|(logo[i]>>3)) ^ magicNum;
}
```

01011101 ^ 11110000
for(int i=0; i<12*2048; i++)
{
    logo[i] = ((logo[i]<<5)|(logo[i]>>3)) ^ magicNum;
}
The key can be either calculated from the Title ID.
COPY PROTECTION

The key can be either calculated from the Title ID

...or guessed by reading any 00 encrypted byte
The key can be either calculated from the Title ID

...or guessed by reading any 00 encrypted byte

00 ^ XX = XX
COPY PROTECTION

The key can be either calculated from the Title ID

...or guessed by reading any 00 encrypted byte

\[00 ^ \text{XX} = \text{XX}\]

The first byte of the logo is always 00
The key can be either calculated from the Title ID

...or guessed by reading any 00 encrypted byte

00 ^ XX = XX

The first byte of the logo is always 00

The 12 first sectors are dedicated to this, The next 2 are for Master Drives, and the last 2 are unused
DRM

COPY PROTECTION
COPY PROTECTION

Unhappy of having encrypted content which you can decrypt by simply reading its first byte Sony added a more convoluted protection mechanism called MagicGate to secure its memory cards.
COPY PROTECTION

Unhappy of having encrypted content which you can decrypt by simply reading its first byte Sony added a more convoluted protection mechanism called MagicGate to secure its memory cards.

You can obviously ask nicely the mechacon to sign and access memory cards for you, but that's not fun.
DRM

MAGICGATE
MagicGate uses DES
MagicGate uses DES

Best public cryptanalysis

DES has been considered insecure right from the start because of the feasibility of brute-force attacks\(^1\). Such attacks have been demonstrated in practice (see EFF DES cracker)
MagicGate uses DES

Best public cryptanalysis
DES has been considered insecure right from the start because of the feasibility of brute-force attacks\textsuperscript{[1]} Such attacks have been demonstrated in practice (see EFF DES cracker)

Oh boy
DRM

MAGICGATE

MagicGate uses 3DES

Best public cryptanalysis

DES has been considered insecure right from the start because of the feasibility of brute-force attacks\textsuperscript{[1]} Such attacks have been demonstrated in practice (see EFF DES cracker)

Oh boy
DRM

MAGICGATE

MagicGate uses 3DES

**Best public cryptanalysis**
Lucks: $2^{32}$ known plaintexts, $2^{113}$ operations including $2^{90}$ DES encryptions, $2^{88}$ memory; Biham: find one of $2^{28}$ target keys with a handful of chosen plaintexts per key and $2^{84}$ encryptions

Oh boy
MagiGate uses 3DES

Best public cryptanalysis
Lucks: $2^{32}$ known plaintexts, $2^{113}$
operations including $2^{90}$ DES
encryptions, $2^{88}$ memory; Biham: find
one of $2^{28}$ target keys with a handful of
chosen plaintexts per key and $2^{84}$
encryptions

...but with only 2 keys of security
Their 3DES implementation changes the key schedule slightly
Their 3DES implementation changes the key schedule slightly. They use it in CBC mode as a challenge reply nonce based cryptosystem.
Their 3DES implementation changes the key schedule slightly

They use it in CBC mode as a challenge reply nonce based cryptosystem

1. We ask the memory card for some IV and its identifier
Their 3DES implementation changes the key schedule slightly. They use it in CBC mode as a challenge reply nonce based cryptosystem.

1. We ask the memory card for some IV and its identifier
2. We derive a unique key based on this
Their 3DES implementation changes the key schedule slightly.

They use it in CBC mode as a challenge reply nonce based cryptosystem.

1. We ask the memory card for some IV and its identifier
2. We derive a unique key based on this
3. We ask the memory card for a nonce it generated
Their 3DES implementation changes the key schedule slightly.

They use it in CBC mode as a challenge reply nonce based cryptosystem.

1. We ask the memory card for some IV and its identifier
2. We derive a unique key based on this
3. We ask the memory card for a nonce it generated
4. We generate our nonce
Their 3DES implementation changes the key schedule slightly. They use it in CBC mode as a challenge reply nonce based cryptosystem.

1. We ask the memory card for some IV and its identifier.
2. We derive a unique key based on this.
3. We ask the memory card for a nonce it generated.
4. We generate our nonce.
5. We generate a challenge: OurNonce|CardNonce|CardIV encrypted with the Unique Key we calculated and using a built-in IV.
Their 3DES implementation changes the key schedule slightly. They use it in CBC mode as a challenge reply nonce based cryptosystem.

1. We ask the memory card for some IV and its identifier.
2. We derive a unique key based on this.
3. We ask the memory card for a nonce it generated.
4. We generate our nonce.
5. We generate a challenge: OurNonce|CardNonce|CardIV encrypted with the Unique Key we calculated and using a built-in IV.
6. The memory card decrypts our challenge and rebuilds another: CardNonce|MechaNonce|SessionKey, using the IV of step 5.
Their 3DES implementation changes the key schedule slightly.

They use it in CBC mode as a challenge reply nonce based cryptosystem.

1. We ask the memory card for some IV and its identifier.
2. We derive a unique key based on this.
3. We ask the memory card for a nonce it generated.
4. We generate our nonce.
5. We generate a challenge: OurNonce|CardNonce|CardIV encrypted with the Unique Key we calculated and using a built-in IV.
6. The memory card decrypts our challenge and rebuilds another: CardNonce|MechaNonce|SessionKey, using the IV of step 5.
7. The SessionKey will now be used as a Key Encryption Key.
This implementation has multiple issues:

- We can pull off chosen plaintext attacks by MITMing the mechacon and the memory card
This implementation has multiple issues:

- We can pull off chosen plaintext attacks by MITMing the mechacon and the memory card
- The IV used for the challenge is baked in and will never change.
This implementation has multiple issues:

- We can pull off chosen plaintext attacks by MITMing the mechacon and the memory card.
- The IV used for the challenge is baked in and will never change.
- We can force Mechacon to keep reusing the same "unique" encryption key by resending the same CardIV and CardID.
This implementation has multiple issues:

- We can pull off chosen plaintext attacks by MITMing the mechacon and the memory card.
- The IV used for the challenge is baked in and will never change.
- We can force Mechacon to keep reusing the same "unique" encryption key by resending the same CardIV and CardID.
- We can arbitrarily replace CardID and CardIV in any communication while keeping the same unique key(!).
This implementation has multiple issues:

- We can pull off chosen plaintext attacks by MITMing the mechacon and the memory card.
- The IV used for the challenge is baked in and will never change.
- We can force Mechacon to keep reusing the same "unique" encryption key by resending the same CardIV and CardID.
- We can arbitrarily replace CardID and CardIV in any communication while keeping the same unique key(!)
- ...not sensible to replay attacks, unlikely to be an oracle.
This implementation has multiple issues:

- We can pull off chosen plaintext attacks by MITMing the mechacon and the memory card.
- The IV used for the challenge is baked in and will never change.
- We can force Mechacon to keep reusing the same "unique" encryption key by resending the same CardIV and CardID.
- We can arbitrarily replace CardID and CardIV in any communication while keeping the same unique key(!)
- ...not sensible to replay attacks, unlikely to be an oracle

Let's be smarter!
MAGICGATE

Mechacon challenge:
Mechacon challenge:
Mechacon challenge:

\[ E_{\text{CardIV}} = E(\text{CardIV} \oplus E_{\text{CardNonce}}, K) \]
Mechacon challenge:

$$E_{CardIV} = E(CardIV \oplus E_{CardNonce}, K)$$

We can always predict everything but K so we can generate infinitely many known plaintext!
We can thus pull off a Linear Cryptanalysis attack on DES with our known plaintext dictionary.
We can thus pull off a Linear Cryptanalysis attack on DES with our known plaintext dictionary.

Matsui's attack can break it using $2^{47}$ plaintext and was published in 1993. MagicGate was published in 1999.
We can thus pull off a Linear Cryptanalysis attack on DES with our known plaintext dictionary. Matsui's attack can break it using $2^{47}$ plaintext and was published in 1993. MagicGate was published in 1999.

Biryukov et al's attack only requires $2^{41}$ and was released in 2004.
We can thus pull off a Linear Cryptanalysis attack on DES with our known plaintext dictionary. Matsui's attack can break it using $2^{47}$ plaintext and was published in 1993. MagicGate was published in 1999. Biryukov et al's attack only requires $2^{41}$ and was released in 2004.

On Multiple Linear Approximations
https://doi.org/10.1007/978-3-540-28628-8_1
hint: sci-hub
We can thus pull off a Linear Cryptanalysis attack on DES with our known plaintext dictionary.

Matsui's attack can break it using $2^{47}$ plaintext and was published in 1993. MagicGate was published in 1999.

Biryukov et al's attack only requires $2^{41}$ and was released in 2004.

On Multiple Linear Approximations

https://doi.org/10.1007/978-3-540-28628-8_1

hint: sci-hub

But this only applies to DES!
Sony uses 3DES with a 2 key scheme, using the two keys on three encryption steps in this order:
Sony uses 3DES with a 2 key scheme, using the two keys on three encryption steps in this order:

\[ k_1, k_2, k_1 \]
Sony uses 3DES with a 2 key scheme, using the two keys on three encryption steps in this order:

$k_1, k_2, k_1$

An incorrect order could make a meet-in-the-middle attack possible, but unfortunately for us no can do here
Sony uses 3DES with a 2 key scheme, using the two keys on three encryption steps in this order:

\[ k_1, k_2, k_1 \]

An incorrect order could make a meet-in-the-middle attack possible, but unfortunately for us no can do here.

Van Oorschot's attack based on Merkle is a known plaintext attack on 3DES with two triples which is now probably achievable by a dedicated adversary.
Sony uses 3DES with a 2 key scheme, using the two keys on three encryption steps in this order:

\[ k_1, k_2, k_1 \]

An incorrect order could make a meet-in-the-middle attack possible, but unfortunately for us no can do here.

Van Oorschot's attack based on Merkle is a known plaintext attack on 3DES with two triples which is now probably achievable by a dedicated adversary.

A known-plaintext attack on two-key triple encryption

https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.66.6575
There are a few other details like Content Keys being derived after that, or the Memory Card replacing the Session Key by its own Storage Key once stored, but they are all vulnerable to this same attack.
There are a few other details like Content Keys being derived after that, or the Memory Card replacing the Session Key by its own Storage Key once stored, but they are all vulnerable to this same attack.

In the end we can extract all keys from mechacon blindly without using nitric acid!

Although I am unsure which is costlier nowadays
There are a few other details like Content Keys being derived after that, or the Memory Card replacing the Session Key by its own Storage Key once stored, but they are all vulnerable to this same attack.

In the end we can extract all keys from mechacon blindly without using nitric acid! Although I am unsure which is costlier nowadays

...Or we can reverse engineer Sony's PS2 emulator which also includes the entire MagicGate algorithm to work with memory card adaptors
There are a few other details like Content Keys being derived after that, or the Memory Card replacing the Session Key by its own Storage Key once stored, but they are all vulnerable to this same attack.

In the end we can extract all keys from mechacon blindly without using nitric acid!
Although I am unsure which is costlier nowadays

...Or we can reverse engineer Sony's PS2 emulator which also includes the entire MagicGate algorithm to work with memory card adaptors

Oops²
Sony, here are some book recommendations if you want to study cryptography/DES a bit more
Sony, here are some book recommendations if you want to study cryptography/DES a bit more

Applied Cryptography, Bruce Schneier
Sony, here are some book recommendations if you want to study cryptography/DES a bit more.

- **Applied Cryptography**, Bruce Schneier
- **Serious Cryptography**, Jean-Philippe Aumasson
Sony, here are some book recommendations if you want to study cryptography/DES a bit more:

- **Applied Cryptography**, Bruce Schneier
- **Serious Cryptography**, Jean-Philippe Aumasson
WHAT IS THE GS
WHAT IS THE GS

A rasterizer
WHAT IS THE GS

A rasterizer

That's it!!!
WHAT IS THE GS

A rasterizer

That's it!!!
WHAT IS THE GS

A rasterizer

That's it!!!

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.0</td>
<td>-1.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>-1.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

 Draws Internally into a Framebuffer. A part of the GS called PCRTC then outputs it to your TV
What is the GS?
WHAT IS THE GS

PREPROCESS → RASTERIZE → BORDER CHECK

PCRTC → TEXTURE MAP
WHAT IS THE GS

PREPROCESS → RASTERIZE → BORDER CHECK

PCRTC → TEXTURE MAP
WHAT IS THE GS

PREPROCESS → RASTERIZE → BORDER CHECK

PCRTC ← TEXURE MAP
What is the GS

1. Preprocess
2. Rasterize
3. Border Check
4. Texture Map
5. PCRTC
And that's a PS2/TV folks!
WHAT IS THE GS
WHAT IS THE GS

Data is transferred to the GS by using the GIF which is a part of the EE
WHAT IS THE GS

Data is transferred to the GS by using the GIF which is a part of the EE

Textures are transferred in a way that pleases the GS pixel units
**WHAT IS THE GS**

Data is transferred to the GS by using the GIF which is a part of the EE.

Textures are transferred in a way that pleases the GS pixel units.

Here is an example with PSMCT32.
WHAT IS THE GS

Data is transferred to the GS by using the GIF which is a part of the EE

Textures are transferred in a way that pleases the GS pixel units

Here is an example with PSMCT32
WHAT IS THE GS
**WHAT IS THE GS**

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
</tr>
</tbody>
</table>
WHAT IS THE GS
WHAT IS THE GS
WHAT IS THE GS
WHAT IS THE GS

4 and 8 bit textures can optionally be indexed and use a palette, which is called in a PS2 world a CLUT (Color LookUp Table)
WHAT IS THE GS

4 and 8 bit textures can optionally be indexed and use a palette, which is called in a PS2 world a CLUT (Color LookUp Table)

Other notable thing: the Framebuffer doesn't have a fixed size and can be resized to 1080p!
WHAT IS THE GS

4 and 8 bit textures can optionally be indexed and use a palette, which is called in a PS2 world a CLUT (Color LookUp Table).

Other notable thing: the Framebuffer doesn't have a fixed size and can be resized to 1080p!

But we cannot output it unfortunately...
WHAT IS THE GS

4 and 8 bit textures can optionally be indexed and use a palette, which is called in a PS2 world a CLUT (Color LookUp Table)

Other notable thing: the Framebuffer doesn't have a fixed size and can be resized to 1080p!

But we cannot output it unfortunately...

...without hacks :D
OTHER HARDWARE
OTHER HARDWARE

GamePads are handled by the IOP. Usually GamePad state is read at each VSync by the game logic.
OTHER HARDWARE

GamePads are handled by the IOP. Usually GamePad state is read at each VSync by the game logic

GamePad communicate with the SIO2 protocol to the PS2, which is an extension of the original PS1 protocol
OTHER HARDWARE

GamePads are handled by the IOP. Usually, GamePad state is read at each VSync by the game logic.

GamePads communicate with the SIO2 protocol to the PS2, which is an extension of the original PS1 protocol.

The IPU is a secondary processor hidden in the EE without any ISA.
OTHER HARDWARE

GamePads are handled by the IOP. Usually GamePad state is read at each VSync by the game logic.

GamePad communicate with the SIO2 protocol to the PS2, which is an extension of the original PS1 protocol.

The IPU is a secondary processor hidden in the EE without any ISA.

You write data, through DMA, send the command and it decodes the stream in real time.
OTHER HARDWARE
OTHER HARDWARE

Syscon is a separate processor on the motherboard that handles power management related tasks

We essentially can forget about it emulation wise
OTHER HARDWARE

Syscon is a separate processor on the motherboard that handles power management related tasks.

We essentially can forget about it emulation wise.

The CDVD subsystem is essentially composed of 3 parts: the laser, a DSP to decode the laser signals and mechacon to ensure DRM.
OTHER HARDWARE

Syscon is a separate processor on the motherboard that handles power management related tasks.

We essentially can forget about it emulation wise.

The CDVD subsystem is essentially composed of 3 parts: the laser, a DSP to decode the laser signals and mechacon to ensure DRM.

The BIOS also has the infamous CSS algorithm to decode DVDs, this is handled by the IOP.
USB and IEEE 1394 are connected to IOP's DMA access
OTHER HARDWARE

USB and IEEE 1394 are connected to IOP's DMA access

The protocols are game specific.
USB and IEEE 1394 are connected to IOP's DMA access

The protocols are game specific.

The SSBUS is essentially the DMA core of the PS2. The EE, IOP, DEV9, CDVD, etc... are all connected to it.
OTHERS - HW

OTHER HARDWARE

USB and IEEE 1394 are connected to IOP's DMA access

The protocols are game specific.

The SSBUS is essentially the DMA core of the PS2. The EE, IOP, DEV9, CDVD, etc... are all connected to it.

DEV9 is a PCMCIA-like device addressed through DMA. Protocols are game specific but are mostly centered around the ethernet and HDD adapter.
A VSync interrupt is reached, EE reads PAD state, transferred through the SIF.
A VSync interrupt is reached, EE reads PAD state, transferred through the SIF
The EE runs the enemy's AI logic, does some trigonometry for hitbox with COP1 and COP2 for the next frame.

```
# kh2ai pseudocode

delta_hitbox = 0.1

if ((keyblade.hitbox - enemy.hitbox) <= delta_hitbox):
    enemy.attack.one_winged_angel()
```
Meanwhile the VU1 calculated the transformations of the 3D model for this frame and transfers it to the GS.
The GS is now ready to draw! Meanwhile the game logic continued and need to play a sound effect.
The GS now draws the frame on screen while the game logic continued and a new 3D model is loaded into VU memory.
And that's how you get a video game!
And that's how you get a video game!

...and diagrams that doesn't make sense
And that's how you get a video game!

...and diagrams that doesn't make sense

The core idea is that the game logic, rendering logic and I/O logic are all able to run in parallel on the different cores
And that's how you get a video game!

...and diagrams that doesn't make sense

The core idea is that the game logic, rendering logic and I/O logic are all able to run in parallel on the different cores

There is an infinite number of possible arrangements of your rendering pipeline, try to imagine others!
HOW DOES EMULATION WORK
What is the first step of making an emulator?
What is the first step of making an emulator?

1 → KH2FM file KH2FM.ISO
2 KH2FM.ISO: UDF filesystem data (version 1.5) ''
3 → KH2FM file SLPM_666.75
4 SLPM_666.75: ELF 32-bit LSB executable, MIPS, MIPS-III version 1 (SYSV), statically linked, stripped
What is the first step of making an emulator?

File parsers!
PARSERS

MEET SYSTEM.CNF
MEET SYSTEM.CNF

```c
int GetPS2ElfName(wxString& name)
{
    int retype = 0;
    try {
        IsoFSCDVD isofs;
        IsoFile file(isofs, L"SYSTEM.CNF;1");
        int size = file.getLength();
        if (size == 0) return 0;
    } [...]
}
```

// return value:
//   0 - Invalid or unknown disc.
//   1 - PS1 CD
//   2 - PS2 CD

PARSERS

MEET SYSTEM.CNF
while (!file.eof()) {
    const wxString original(fromUTF8(file.readLine().c_str()));
    const ParsedAssignmentString parts(original);
    if (parts.lvalue.IsEmpty() && parts.rvalue.IsEmpty()) continue;
    if (parts.rvalue.IsEmpty() && file.getLength() != file.getSeekPos()) {
        Console.Warning("(SYSTEM.CNF) Unusual or malformed entry in SYSTEM.CNF ignored:");
        Console.Indent().WriteLn(original);
        continue;
    }
}

PARSERS

MEET SYSTEM.CNF
if ( parts.lvalue == L"BOOT2" )
{
    name = parts.rvalue;
    Console.WriteLine( Color_StrongBlue, L"(SYSTEM.CNF) Detected PS2 Disc = " + name );
    retype = 2;
}
else if ( parts.lvalue == L"BOOT" )
{
    name = parts.rvalue;
    Console.WriteLine( Color_StrongBlue, L"(SYSTEM.CNF) Detected PSX/PSone Disc = " + name );
    retype = 1;
}
else if ( parts.lvalue == L"VMODE" )
{
    Console.WriteLine( Color_Blue, L"(SYSTEM.CNF) Disc region type = " + parts.rvalue );
}
else if ( parts.lvalue == L"VER" )
{
    Console.WriteLine( Color_Blue, L"(SYSTEM.CNF) Software version = " + parts.rvalue );
}
void R5900::Interpreter::OpcodeImpl::SWC1() {
  u32 addr;
  // force sign extension to 32bit
  addr = cpuRegs.GPR.r[_Rs_].UL[0] + (s16)(cpuRegs.code & 0xffff);
  if (addr & 0x00000003)
    { Console.Error("FPU (SWC1 Opcode): Invalid Unaligned Memory Address");
      return;
    } // Should signal an exception?
  memWrite32(addr, fpuRegs.fpr[_Rt_].UL);
}

void recSWC1()
{ recCall(::R5900::Interpreter::OpcodeImpl::SWC1); }
void recSWC1()
{
    #ifndef FPU_RECOMPILE
        recCall(::R5900::Interpreter::OpcodeImpl::SWC1);
    #else
        _deleteFPtoXMMreg(_Rt_, 1);
        xMOV(arg2regd, ptr32[&fpuRegs.fpr[_Rt_].UL]);
        if( GPR_IS_CONST1(_Rs_) )
        {
            int addr = g_cpuConstRegs[_Rs_].UL[0] + _Imm_;
            vtlb_DynGenWrite_Const(32, addr);
        }
        else
        {
            _eeMoveGPRtoR(arg1regd, _Rs_);
            if (_Imm_ != 0)
                xADD(arg1regd, _Imm_);
            iFlushCall(FLUSH_FULLVTLB);
            vtlb_DynGenWrite(32);
        }
    #endif
}

EE::Profiler.EmitOp(eeOpcode::SWC1);
void mmap_PageFaultHandler::OnPageFaultEvent( const PageFaultInfo& info, bool& handled )
{
    pxAssert( eeMem );

    // get bad virtual address
    uptr offset = info.addr - (uptr)eeMem->Main;
    if( offset >= Ps2MemSize::MainRam ) return;

    mmap_ClearCpuBlock( offset );
    handled = true;
}

// offset - offset of address relative to psM.
// All recompiled blocks belonging to the page are cleared, and any new blocks recompiled
// from code residing in this page will use manual protection.
static __fi void mmap_ClearCpuBlock( uint offset )
{
    [...]
BIOS

SYSCALL
SYSCALL

We can sorta emulate some instructions!
SYSCALL

We can sorta emulate some instructions!

We now need to emulate PS2-specific ones

2 ways to do it:
BIOS

SYSCALL

We can sorta emulate some instructions!

We now need to emulate PS2-specific ones

  2 ways to do it:

  • HLE: Reimplement them like an interpreter
SYSCALL

We can sorta emulate some instructions!

We now need to emulate PS2-specific ones

2 ways to do it:

- HLE: Reimplement them like an interpreter
- LLE: Run the BIOS
SYSCALL

We can sorta emulate some instructions!

We now need to emulate PS2-specific ones

2 ways to do it:

- **HLE**: Reimplement them like an interpreter
- **LLE**: Run the BIOS
SYSCALL

We can sorta emulate some instructions!

We now need to emulate PS2-specific ones

2 ways to do it:
- HLE: Reimplement them like an interpreter
- LLE: Run the BIOS

PS2 GAMES PATCH THE BIOS
DUMPING THE BIOS
DUMPING THE BIOS

The BIOS is available on the flash chip!
DUMPING THE BIOS

The BIOS is available on the flash chip!

Unencrypted!!
The BIOS is available on the flash chip!

Unencrypted!!

Save for the DVD EROM, probably to hide the CSS
The BIOS is available on the flash chip!

Unencrypted!!

Save for the DVD EROM, probably to hide the CSS

We don't really care about it though :D
DUMPING THE BIOS

The BIOS is available on the flash chip!

Unencrypted!!

Save for the DVD EROM, probably to hide the CSS

We don't really care about it though :D

A few soldering hackjobs later...
BIOS ENTRYPONT

1  mfc0   k0,PRId ; get register PRId from COP0
2    nop
3  slti   at,k0,0x59 ; if (0x59<=k0) at = 0
4  bne    at,zero,LAB_00000024 ; if (at == 0) jmp LAB_00000024
5    nop
BIOS ENTRYPUNCT

1 mfc0 k0,PRId ; get register PRId from COP0
2 nop
3 slti at,k0,0x59 ; if (0x59<=k0) at = 0
4 bne at,zero,LAB_00000024 ; if (at == 0) jmp LAB_00000024
5 nop
BIOS ENTRYPONIT

1  mfc0  k0,PRId ; get register PRId from COP0
2    nop
3  slti  at,k0,0x59 ; if (0x59<=k0) at = 0
4  bne  at,zero,LAB_00000024 ; if (at == 0) jmp LAB_00000024
5    nop
BIOS ENTRYPONIT

```assembly
1 mfc0 k0,PRId ; get register PRId from COP0
2 nop
3 slti at,k0,0x59 ; if (0x59<=k0) at = 0
4 bne at,zero,LAB_00000024 ; if (at == 0) jmp LAB_00000024
5 nop
```

COP0 is not the same between the IOP and the EE
BIOS ENTRYPOINT

COP0 is not the same between the IOP and the EE

This bit of code effectively is the entrypoint for both the IOP and the EE

```
1 mfc0 k0,PRId ; get register PRId from COP0
2 nop
3 slti at,k0,0x59 ; if (0x59<=k0) at = 0
4 bne at,zero,LAB_00000024 ; if (at == 0) jmp LAB_00000024
5 nop
```
COP0 is not the same between the IOP and the EE

This bit of code effectively is the entrypoint for both the IOP and the EE

We already have to emulate the IOP
CUSTOM ARCHITECTURE
CUSTOM ARCHITECTURE

How do you figure out a custom ISA?
CUSTOM ARCHITECTURE

How do you figure out a custom ISA?

Essentially 2 ways:
CUSTOM ARCHITECTURE

How do you figure out a custom ISA?

Essentially 2 ways:

• Make assumptions, test assumptions on hardware
CUSTOM ARCHITECTURE

How do you figure out a custom ISA?

Essentially 2 ways:

- Make assumptions, test assumptions on hardware
- Get documentation from Sony
CUSTOM ARCHITECTURE

How do you figure out a custom ISA?

Essentially 2 ways:

• Make assumptions, test assumptions on hardware
• Get documentation from Sony
CUSTOM ARCHITECTURE

How do you figure out a custom ISA?

Essentially 2 ways:

• Make assumptions, test assumptions on hardware
• Get documentation from Sony

Here's a talk for some insight on the process:
CUSTOM ARCHITECTURE

How do you figure out a custom ISA?

Essentially 2 ways:

- Make assumptions, test assumptions on hardware
- Get documentation from Sony

Here's a talk for some insight on the process:

Reverse engineering of binary programs for custom virtual machines
MMU

Memory

Virtual Memory
The EE has an MMU we need to emulate, meet VTLB!

```c
void __fastcall vtlb_memRead64(u32 mem, mem64_t *out) {
    auto vmv = vtlbdata.vmap[mem>>VTLB_PAGE_BITS];
    if (!vmv.isHandler(mem)) {
        if (!CHECK_EERE) {
            if (CHECK_CACHE && CheckCache(mem)) {
                *out = readCache64(mem);
                return;
            }
        }
        *out = *(mem64_t*)vmv.assumePtr(mem);
    } [...]
}
The EE has an MMU we need to emulate, meet recVTLB!

```c
// TLB lookup is performed in const, with the assumption that the COP0/TLB will clear the
// recompiler if the TLB is changed.
void vtlb_DynGenRead64_Const( u32 bits, u32 addr_const )
{
    EE::Profiler.EmitConstMem(addr_const);

    auto vmv = vtlbdata.vmap[addr_const>>VTLB_PAGE_BITS];
    if( !vmv.isHandler(addr_const) )
    {
        auto ppf = vmv.assumePtr(addr_const);
        switch( bits )
        {
            case 64:
                iMOV64_Smart( ptr[arg2reg], ptr[(void*)ppf] );
                break;
            case 128:
                iMOV128_SSE( ptr[arg2reg], ptr[(void*)ppf] );
                break;
            jNO_DEFAULT
        }
    } [...]
```
MULTI CORE SHENANIGANS
MULTI CORE SHENANIGANS

Now that we have multiple CPU cores running in parallel we need to handle them concurrently.
MULTI CORE SHENANIGANS

Now that we have multiple CPU cores running in parallel we need to handle them concurrently.

We have our own thread scheduler to do that, meet SysExecutor!
void pxEvtQueue::ProcessEvent(SysExecEvent* evt)
{
    if (!evt) return;

    if (wxThread::GetCurrentId() != m_OwnerThreadId)
    {
        SynchronousActionState sync;
        evt->SetSyncState(sync);
        PostEvent(evt);
        sync.WaitForResult();
    }

    else
    {
        std::unique_ptr<SysExecEvent> deleteMe(evt);
        deleteMe->_DoInvokeEvent();
    }

    [...]
}
DISPATCHING TO PROCESSORS
How do we transfer data from, say, the IOP to DEV9?
How do we transfer data from, say, the IOP to DEV9?

Our JIT fallbacks to Interpreters and verifies where the write should go!
static void rpsxSB()
{
    _psxDeleteReg(_Rs_, 1);
    _psxDeleteReg(_Rt_, 1);

    xMOV(arg1regd, ptr32[&psxRegs.GPR.r[_Rs_]])
    if (_Imm_) xADD(arg1regd, _Imm_);
    xMOV(  arg2regd,  ptr32[&psxRegs.GPR.r[_Rt_]]  );
    xFastCall((void*)iopMemWrite8, arg1regd, arg2regd );
}

void __fastcall iopMemWrite8(u32 mem, u8 value)
{
    mem &= 0x1fffffff;
    u32 t = mem >> 16;
    [...]
    else
    {
        if (!(p != NULL && !(psxRegs.CP0.n.Status & 0x10000)))
        {
            if (t == 0x1000)
            {
                DEV9write8(mem, value); return;
            }
        }
        PSXMEM_LOG("err sb %8.8lx = %x", mem, value);
    }
}
EMULATING SOUND

SPU2
EMULATING SOUND

We run an async loop that processes audio while everything else is running.
```c
__forceinline void TimeUpdate(u32 cClocks)
{
    u32 dClocks = cClocks - lClocks;

    // Sanity Checks:
    // It's not totally uncommon for the IOP's
    // clock to jump backwards a cycle or two, and in
    // such cases we just want to ignore the TimeUpdate call.
    if (dClocks > (u32)-15)
        return;

    if (SynchMode == 1) // AsyncMix on
        SndBuffer::UpdateTempoChangeAsyncMixing();
    else
        TickInterval = 768; // Reset to default

    // Update Mixing Progress
    while (dClocks >= TickInterval)
    {
        for (int i = 0; i < 2; i++)
        {
            if (has_to_call_irq[i])
            {
                has_to_call_irq[i] = false;
                if (!(!Spdif.Info & (4 << i)) & Cores[i].IRQEnable)
                    { Spdif.Info |= (4 << i);
                      if (!SPU2_dummy_callback)
                          spu2Irq();
                      Mix();
                      [...]
                    } } } 
    Mix();
[...]
} 
```
__forceinline void Mix() {
    [...] 
    Out.Left *= FinalVolume;
    Out.Right *= FinalVolume;
    SndBuffer::Write(Out);
    [...] 
}

void SndBuffer::Write(const StereoOut32& Sample) {
    [...] 
    else {
        if (SynchMode == 0) // TimeStretch on
            timeStretchWrite();
        else
            _WriteSamples(sndTempBuffer, SndOutPacketSize);
    }
}

void SndOut_SDL::callback_fillBuffer(void* userdata, Uint8* stream, int len) {
    [...] 
    for (Uint16 i = 0; i < sdl_samples; i += SndOutPacketSize)
        SndBuffer::ReadSamples(&buffer[i]);
    SDL_MixAudio(stream, (Uint8*)buffer.get(), len, SDL_MIX_MAXVOLUME);
}
void GSState::FlushPrim()
{
    if (m_index.tail > 0)
    {
        GL_REG("FlushPrim ctxt %d", PRIM->CTXT);
        [...]
        if (GSLocalMemory::m_psm[m_context->FRAME.PSM].fmt < 3 && GSLocalMemory::m_psm[m_context->ZBUF.PSM].fmt < 3)
        {
            m_vt.Update(m_vertex.buff, m_index.buff, m_vertex.tail, m_index.tail, GSUtil::GetPrimClass(PRIM->PRIM));
            m_context->SaveReg();
            try
            {
                Draw();
            }
        }
    }
}
void GSRendererHW::Draw()
{
    if (m_dev->IsLost() || IsBadFrame()) {
        GL_INS(“Warning skipping a draw call (%d)”, s_n);
        return;
    }
    GL_PUSH(“HW Draw %d”, s_n);
    [...
    GSTextureCache::Target* rt = NULL;
    GSTexture* rt_tex = NULL;
    if (!no_rt) {
        rt = m_tc->LookupTarget(TEX0, m_width, m_height, GSTextureCache::RenderTarget, true, fm);
        rt_tex = rt->m_texture;
    }
    TEX0.TBP0 = context->ZBUF.Block();
    TEX0.TBW = context->FRAME.FBW;
    TEX0.PSM = context->ZBUF.PSM;
    GSTextureCache::Target* ds = NULL;
    GSTexture* ds_tex = NULL;
    if (!no_ds) {
        ds = m_tc->LookupTarget(TEX0, m_width, m_height, GSTextureCache::DepthStencil, context->DepthWrite());
        ds_tex = ds->m_texture;
    }
    [...
    DrawPrims(rt_tex, ds_tex, m_src);
void GSRendererOGL::DrawPrims(GSTexture* rt, GSTexture* ds, GSTextureCache::Source* tex)
{
    // HLE implementation of the channel selection effect
    // Warning it must be done at the beginning because it will change the
    // vertex list (it will interact with PrimitiveOverlap and accurate
    // blending)
    EmulateChannelShuffle(&rt, tex);
    // Upscaling hack to avoid various line/grid issues
    MergeSprite(tex);
    // Always check if primitive overlap as it is used in plenty of effects.
    m_prim_overlap = PrimitiveOverlap();
    // Blend if (!IsOpaque() && rt) {
        EmulateBlending(DATE_GL42, DATE_GL45);
    } else {
        dev->OMSetBlendState(); // No blending please
    }
    if (m_ps_sel.dfmt == 1) {
        // Disable writing of the alpha channel
        m_om_csel.wa = 0;
    }
    if (DATE && !DATE_GL45) {
        GSVector4i dRect = ComputeBoundingBox(rtscale, rtsize);
    }
    dev->BeginScene();
    EmulateZbuffer(); // will update VS depth mask
PAD, DEV9, USB, MCD and CDVD works relatively similarly and as such I won't mention them for brevity sake.
OTHER COMPONENTS

PAD, DEV9, USB, MCD and CDVD work relatively similarly and as such I won't mention them for brevity sake.

Memory writes are handled by the module, which simulates the I/O.
OTHER COMPONENTS

PAD, DEV9, USB, MCD and CDVD works relatively similarly and as such I won't mention them for brevity sake.

Memory writes are handled by the module, which simulates the I/O.

It is then piped to one of multiple system backend. e.g.:
OTHER COMPONENTS

PAD, DEV9, USB, MCD and CDVD works relatively similarly and as such I won't mention them for brevity sake.

Memory writes are handled by the module, which simulates the I/O.

It is then piped to one of multiple system backend. e.g.:

- PAD: SDL
PAD, DEV9, USB, MCD and CDVD works relatively similarly and as such I won't mention them for brevity sake.

Memory writes are handled by the module, which simulates the I/O.

It is then piped to one of multiple system backend. e.g.:

- PAD: SDL
- DEV9: TAP
OTHER COMPONENTS

PAD, DEV9, USB, MCD and CDVD works relatively similarly and as such I won't mention them for brevity sake.

Memory writes are handled by the module, which simulates the I/O.

It is then piped to one of multiple system backend. e.g.:

- PAD: SDL
- DEV9: TAP
- USB-video: V4L
PAD, DEV9, USB, MCD and CDVD works relatively similarly and as such I won't mention them for brevity sake.

Memory writes are handled by the module, which simulates the I/O.

It is then piped to one of multiple system backend. e.g.:

- PAD: SDL
- DEV9: TAP
- USB-video: V4L
- MCD: your filesystem!
OTHER COMPONENTS

PAD, DEV9, USB, MCD and CDVD work relatively similarly and as such I won't mention them for brevity sake.

Memory writes are handled by the module, which simulates the I/O.

It is then piped to one of multiple system backend. e.g.:

- PAD: SDL
- DEV9: TAP
- USB-video: V4L
- MCD: your filesystem!
- CDVD: <linux/cdrom.h>
WHAT’S LEFT?
WHAT’S LEFT?

Threading the GS and the VU!
WHAT’S LEFT?

Threading the GS and the VU!

Threading the GS is done by waiting for data to be received then have multiple rendering threads in parallel when all transfers are achieved.
WHAT’S LEFT?

Threading the GS and the VU!

Threading the GS is done by waiting for data to be received then have multiple rendering threads in parallel when all transfers are achieved.

Fairly safe
WHAT’S LEFT?

Threading the GS and the VU!

Threading the GS is done by waiting for data to be received then have multiple rendering threads in parallel when all transfers are achieved

Fairly safe

Threading the VU is much harder and not nearly as safe
WHAT’S LEFT?

Threading the GS and the VU!

Threading the GS is done by waiting for data to be received then have multiple rendering threads in parallel when all transfers are achieved

Fairly safe

Threading the VU is much harder and not nearly as safe

Still considered a SpeedHack, still break things
WHAT’S LEFT?

Threading the GS and the VU!

Threading the GS is done by waiting for data to be received then have multiple rendering threads in parallel when all transfers are achieved

Fairly safe

Threading the VU is much harder and not nearly as safe

Still considered a SpeedHack, still break things

Read up our dev blog about threading VU1 for more infos!
WHAT’S LEFT?
WHAT’S LEFT?

Not going too fast!!

Changes by Refraction:
Implement basic cycle counting for COP2 operations, implement COP2 detection while not interlocking, implement Mbit, change drastically cycles required to run every microprogram. Improved flag handling while COP2 update them.

Additionally:
As explained here: Link. Hardware tests proved that VU run at the same speed as EE mips core. So lets set that in pcsx2 where it is possible.
WHAT'S LEFT?

Fixed games in this commit:

- (VIF) Hitman games - Could have potentially crashed randomly with TLB misses or FIFO errors, no longer happening
- 24 The game, Primal, Ghosthunter - No longer need patches to get full speed
- Air Rescue Ranger - Textures are now displayed correctly
- Amplitude - SPS on characters fixed
- Gift, Woody Woodpecker, Kaan - Now work full speed
- Lotus Challenge - Cars are no longer bouncy!
- My Street - missing characters now visible, still exhibit a small amount of SPS in microVu0 but perfect in Vu0 int
- Mike Tysons Heavyweight Box - T posing see-through characters are now whole and animated
- Next Generation Tennis 2003 - No longer need patch to fix SPS
- Nihonbaka Pro Yakkyuu: Final League / World Fantasista - Random glitches are gone
- Phase Paradox - Lighting and Camera in cutscenes are fixed
- Rayman 2 Revolution - Random jittering no longer happens

Game issues fixed:
fixes #1448 fixes #3252 fixes #3028 fixes #1473 fixes #94

- Sega Superstars Tennis - SPS on hands/feet is now gone
- Tiger Woods PGA Tour 2002 - Fixed player stance
- Tony Hawk 4 - Wakeboarding Unleashed demo no longer crash at loading screen (demo need XGKick hack)
- Totally spies Totally Party! - Bad SPS somewhat fixed - Will require you to set EE Cyclerate + 3 to completely fix.
- Twisted Metal Head-On - Black doors have now proper colors
- Wakeboarding Unleashed - No longer hangs getting to the menu on release builds
- World Series Baseball 2k3 - No longer hang on loading screen (game still have other issues)
WHAT’S LEFT?

Faster isn't always better!

Fixed games in this commit:

- (VIF) Hitman games - Could have potentially crashed randomly with TLB misses or FIFO errors, no longer happening
- 24 The game, Primal, Ghosthunter - No longer need patches to get full speed
- Air Rescue Ranger - Textures are now displayed correctly
- Amplitude - SPS on characters fixed
- Gift, Woody Woodpecker, Kaan - Now work full speed
- Lotus Challenge - Cars are no longer bouncy!
- My Street - missing characters now visible, still exhibit a small amount of SPS in microVu0 but perfect in VU0 Int
- Mike Tyson’s Heavyweight Box - T posing see-through characters are now whole and animated
- Next Generation Tennis 2003 - No longer need patch to fix SPS
- Nichibei kan Pro Yakyuu: Final League / World Fantasista - Random glitches are gone
- Phase Paradox - Lighting and camera in cutscenes are fixed
- Rayman 2 Revolution - Random jittering no longer happens

Game issues fixed:
fixes #1448 fixes #3252 fixes #3028 fixes #1473 fixes #94

- Sega Superstars Tennis - SPS on hands/feet is now gone
- Tiger woods PGA Tour 2002 - Fixed player stance
- Tony Hawk 4 - Wakeboarding Unleashed demo no longer crash at loading screen (demo need XGKick hack)
- Totally Spies Totally Party! - Bad SPS somewhat fixed - Will require you to set EE Cyclerate + 3 to completely fix.
- Twisted Metal Head-On - Black doors have now proper colors
- Wakeboarding Unleashed - No longer hangs getting to the menu on release builds
- World Series Baseball 2k3 - No longer hang on loading screen (game still have other issues)
WHAT’S LEFT?
WHAT’S LEFT?
Emulating the laws of physics
WHAT’S LEFT?

Emulating the laws of physics

No, really
WHAT'S LEFT?

Emulating the laws of physics

No, really
WHAT'S LEFT?

Emulating the laws of physics

No, really

```
1 // Read speed is roughly 37% at lowest and full speed on outer edge.
2 // I imagine it's more logarithmic than this
3 // Required for Shadowman to work
4 // Use SeekToSector as Sector hasn't been updated yet
5 const float sectorSpeed = (((float)(cdvd.SeekToSector-offset) / numSectors) * 0.63f) + 0.37f;
6 //DevCon.Warning("Read speed %f sector %d\n", sectorSpeed, cdvd.Sector);
7 return ((PSXCLK * cdvd.BlockSize) / ((float)(((mode == MODE_CDROM) ?
8         PSX_CD_READSPEED : PSX_DVD_READSPEED) * cdvd.Speed) * sectorSpeed));
```
WHAT’S LEFT?
WHAT’S LEFT?

Making an infrastructure!
WHAT’S LEFT?

Making an infrastructure!

A website, forum, compatibility list, get testers...
WHAT’S LEFT?

Making an infrastructure!

A website, forum, compatibility list, get testers...

This is where **YOU** come in :D
WHAT’S LEFT?

Making an infrastructure!
A website, forum, compatibility list, get testers...

This is where **YOU** come in :D

We always need help, feel free to hang out and say hi!
WHAT’S LEFT?

Making an infrastructure!

A website, forum, compatibility list, get testers...

This is where **YOU** come in :D

We always need help, feel free to hang out and say hi!

https://discord.com/invite/TCz3t9k
WHAT'S LEFT?

Making an infrastructure!
A website, forum, compatibility list, get testers...

This is where **YOU** come in :D

We always need help, feel free to hang out and say hi!

https://discord.com/invite/TCz3t9k

You can bridge it to matrix with
https://github.com/matrix-discord/mx-puppet-discord
STATE OF THE PROJECT
STATE OF THE PROJECT

PCSX2 is really old
STATE OF THE PROJECT

PCSX2 is really old

It has now a lot of legacy code that simply needs to be redone, redesigned or freshened up
PCSX2 is really old

It has now a lot of legacy code that simply needs to be redone, redesigned or freshened up

I am leading a whole codebase redesign effort
PCSX2 is really old

It has now a lot of legacy code that simply needs to be redone, redesigned or freshened up

I am leading a whole codebase redesign effort

I'll show you in the next slides the state of things and what to expect!
CODE ARCHITECTURE

- SysExecutor
- MCD
- GUI
- EE
CODE ARCHITECTURE

Plugins

GS  SPU2

Core

GUI  EE

et cetera

Utilities

LibEmitter

et cetera
CODE ARCHITECTURE

SysExecutor

Plugins

GS  SPU2  USB  CDVD  PAD  DEV9

Utilities

LibEmitter  et cetera

Core

MCD

GUI  EE  et cetera
CODE ARCHITECTURE

Plugins

Stubbed
FW
GS
SPU2
USB
CDVD
PAD
DEV9

Utilities
LibEmitter
et cetera

Core
Built-in Plugin
MCD
GUI
EE
et cetera

et cetera
CODE ARCHITECTURE

Hard dependency on wxWidgets!!

SysExecutor

Plugins

Stubbed
FW
GS
SPU2
USB
CDVD
PAD
DEV9

Utilities
LibEmitter
et cetera

Core
Built-in Plugin
MCD
GUI
EE
et cetera
JIT
With all the mentioned challenges, it will take a couple of months to get things working reasonably stable. By that time, more people would have switched to 64bit OSs. If we're even half right in our estimates, Pcsx2 will run much faster on a 64bit OS than on a 32bit OS on the same computer once x86-64 recompilation is done.
With all the mentioned challenges, it will take a couple of months to get things working reasonably stable. By that time, more people would have switched to 64bit OSs. If we're even half right in our estimates, Pcsx2 will run much faster on a 64bit OS than on a 32bit OS on the same computer once x86-64 recompilation is done.

**PCSX2 64bit Recompilation**

*Created: 29 October 2006  Written by ZeroFrog*
With all the mentioned challenges, it will take a couple of months to get things working reasonably stable. By that time, more people would have switched to 64bit OSs. If we're even half right in our estimates, Pcsx2 will run much faster on a 64bit OS than on a 32bit OS on the same computer once x86-64 recompilation is done.

**PCSX2 64bit Recompilation**

*Created: 29 October 2006  Written by ZeroFrog*
JIT
Fortunately, our 64 bit JIT is mostly done!
Fortunately, our 64 bit JIT is mostly done!

The memory card stuff should be sorted now, so Linux at least is ready to go. For GSdx, I understand tellow has it mostly working and @Govanify is planning to take a closer look. Will need to sort out mipmaping but after those two we shall be ready for public release (and any fun bugs that come).
Fortunately, our 64 bit JIT is mostly done!

Unfortunately I'm giving a talk instead of fixing it :)

The memory card stuff should be sorted now, so Linux at least is ready to go. For GSdx, I understand tellow has it mostly working and @GovaniFY is planning to take a closer look. Will need to sort out mipmaping but after those two we shall be ready for public release (and any fun bugs that come).
I've worked on a new protocol for
3 way game<->emulator<-> OS communication
I've worked on a new protocol for 3 way game<->emulator<->OS communication.

A 3-way communication can thus be established, game->OS; OS->game; OS->emu and game->emu.

Game->OS IPC is poll based, OS->game event, game->emu poll and OS->emu event.

This is due to how the IPC is implemented: Dolphin IPC implements an event based IPC for emu<->game at the cost of having to modify the executable code of the game to implement this.

The upside of this PR is thus that you do not need to modify the game executable, only to reverse engineer it to find state variables and read off from it. As such this is not a no-cost implementation if the logic requires this.
I've worked on a new protocol for 3 way game<->emulator<->OS communication.

Romhacks and game modding tools are about to get a lot more interesting!
STATE OF THE PROJECT
STATE OF THE PROJECT

What to expect for 1.8:
STATE OF THE PROJECT

What to expect for 1.8:

- No Plugins!
STATE OF THE PROJECT

What to expect for 1.8:

- No Plugins!
- 64-bit Support!
STATE OF THE PROJECT

What to expect for 1.8:

- No Plugins!
- 64-bit Support!
- Reduced Input Lag!
STATE

STATE OF THE PROJECT

What to expect for 1.8:

- No Plugins!
- 64-bit Support!
- Reduced Input Lag!
- A new shiny IPC protocol!
STATE OF THE PROJECT

What to expect for 1.8:

- No Plugins!
- 64-bit Support!
- Reduced Input Lag!
- A new shiny IPC protocol!
- ...and much more (read our progress reports!)
STATE

STATE OF THE PROJECT
STATE OF THE PROJECT

What might be ready for 2.0:

• A New Qt based GUI along with support for pluggable & community GUIs
STATE OF THE PROJECT

What might be ready for 2.0:

- A New Qt based GUI along with support for pluggable & community GUIs
- Rework of our Infrastructure/Website
STATE OF THE PROJECT

What might be ready for 2.0:

- A New Qt based GUI along with support for pluggable & community GUIs
- Rework of our Infrastructure/Website
- Work on a pluggable JIT backend
STATE OF THE PROJECT

What might be ready for 2.0:

• A New Qt based GUI along with support for pluggable & community GUIs
• Rework of our Infrastructure/Website
• Work on a pluggable JIT backend
• A full cleanup of the codebase!
STATE OF THE PROJECT

What might be ready for 2.0:

- A New Qt based GUI along with support for pluggable & community GUIs
- Rework of our Infrastructure/Website
- Work on a pluggable JIT backend
- A full cleanup of the codebase!
- And hopefully other nice surprises ;)
CLOSING NOTES
CLOSING NOTES

We do not care about emulation wars
CLOSING NOTES

We do not care about emulation wars

It's always a tradeoff, we chose playability over accuracy (we still aim for accuracy)
CLOSING NOTES

We do not care about emulation wars

It's always a tradeoff, we chose playability over accuracy (we still aim for accuracy)

We all have different problems and different solutions
CLOSING NOTES

We do not care about emulation wars

It's always a tradeoff, we chose playability over accuracy (we still aim for accuracy)

We all have different problems and different solutions

Come hang out with us, chill and have fun, that's what emulation is all about!
CLOSING NOTES

We do not care about emulation wars

It's always a tradeoff, we chose playability over accuracy (we still aim for accuracy)

We all have different problems and different solutions

Come hang out with us, chill and have fun, that's what emulation is all about!

If you don't have fun, why even work on a project that you know you won't ever be paid for?
THANKS
THANKS

PCSX2 Team:
- refraction
- kotjin
- TellowKrinkle
- LightningTerror
- arcum42
- bositman
- jackun
- And others including past members like air and cottonvibes!
THANKS

Friends:
- sirocyk
- ellie

PCSX2 Team:
- refraction
- kotjin
- TellowKrinkle
- LightningTerror
- arcum42
- bositman
- jackun
- And others including past members like air
  and cottonvibes!
THANKS

Friends:
- sirocyl
- ellie

The PCSX2 Community:
- CK1
- Vaser
- RedDevilus
- pandubz

PCSX2 Team:
- refraction
- kotjin
- TellowKrinkle
- LightningTerror
- arcum42
- bositman
- jackun
- And others including past members like air and cottonvibes!
THANKS

Friends:
- sirocyl
- ellie

The PCSX2 Community:
- CK1
- Vaser
- RedDevilus
- pandubz

Our Users

PCSX2 Team:
- refraction
- kotjin
- TellowKrinkle
- LightningTerror
- arcum42
- bositman
- jackun
- And others including past members like air and cottonvibes!
THANKS

Friends:
- sirocyl
- ellie

The PCSX2 Community:
- CK1
- Vaser
- RedDevilus
- pandubz

Our Users

...And everyone else I forgot!

PCSX2 Team:
- refraction
- kotjin
- TellowKrinkle
- LightningTerror
- arcum42
- bositman
- jackun
- And others including past members like air and cottonvibes!
THANK YOU!

@GovanifY
govanify.com
gauvain@govanify.com

@PCSX2
pcsx2.net
info@pcsx2.net