Extension: Assembly

Specifications

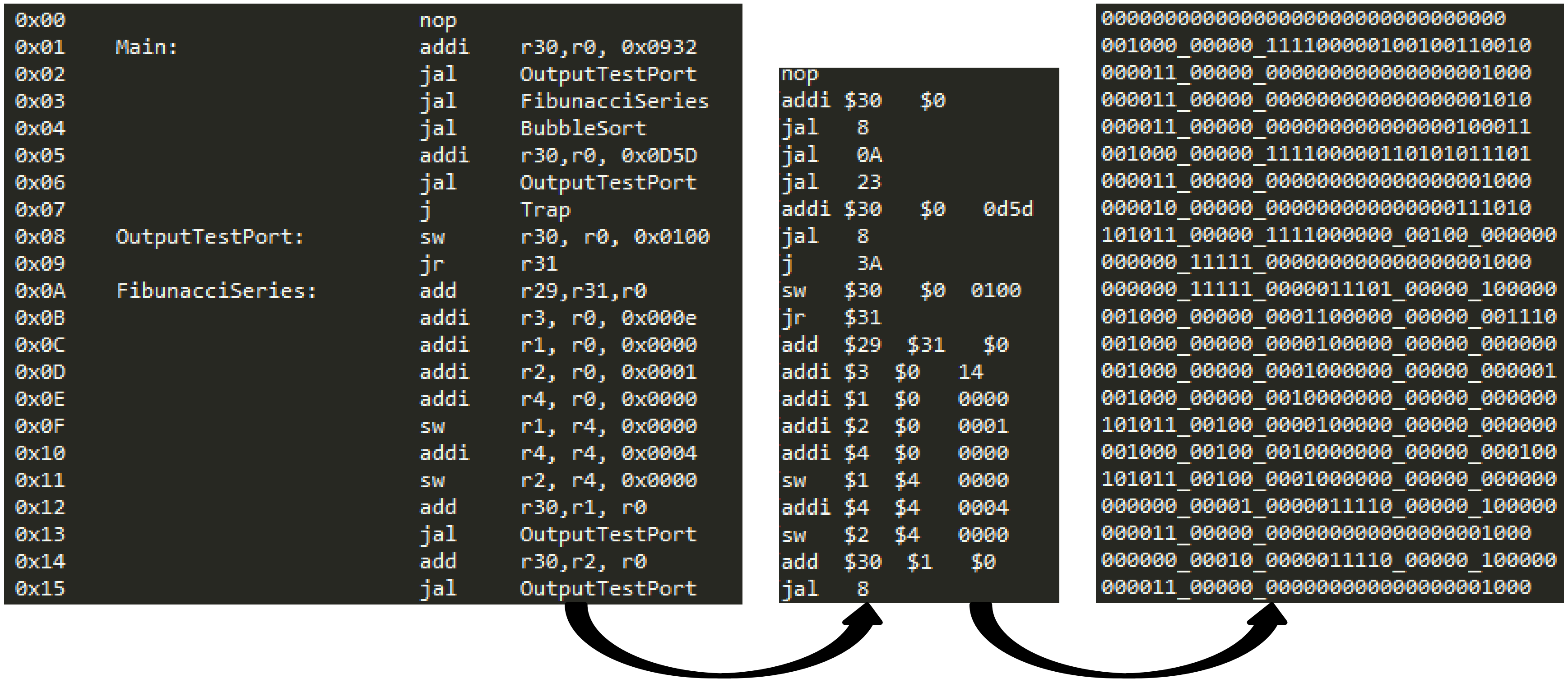
Extension Description

- Main Goal
  - From assembly language to MIPS processor
    - Convert assembly code in SPIM to machine code
    - Execute the machine code on design processor
  - Hardware/Software co-design
    - **For improving performance,**
    - **What unit need needed for processor according to the code**
    - **Based on the processor, how to modify the code**
  - Assembly code: **Bubble sort**
    - **Length sequence: 500**
    - Better to compare with other sorting algorithm
  - **+define+Assembly** in ncverilog simulation command

Comparison Metrics

- Base on the test program
  - “I_mem_Assembly”
- Score : Total execution cycles of I_mem_Assembly

From Assembly to Machine Code



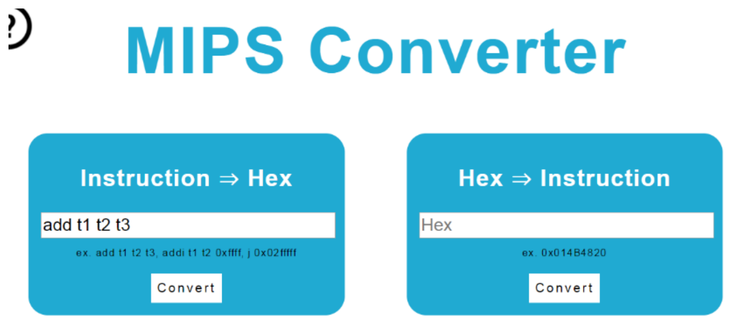
```
0x00                            nop
0x01    Main:                   addi    r30,r0, 0x0932
0x02                            jal     OutputTestPort
0x03                            jal     FibunacciSeries
0x04                            jal     BubbleSort
0x05                            addi    r30,r0, 0x0D5D
0x06                            jal     OutputTestPort
0x07                            j       Trap
0x08    OutputTestPort:         sw      r30, r0, 0x0100
0x09                            jr      r31
0x0A    FibunacciSeries:        add     r29,r31,r0
0x0B                            addi    r3, r0, 0x000e
0x0C                            addi    r1, r0, 0x0000
0x0D                            addi    r2, r0, 0x0001
0x0E                            addi    r4, r0, 0x0000
0x0F                            sw      r1, r4, 0x0000
0x10                            addi    r4, r4, 0x0004
0x11                            sw      r2, r4, 0x0000
0x12                            add     r30,r1, r0
0x13                            jal     OutputTestPort
0x14                            add     r30,r2, r0
0x15                            jal     OutputTestPort
```

```
nop
addi $30   $0
jal   8
jal   0A
jal   23
addi $30   $0   0d5d
jal   8
j     3A
sw   $30   $0  0100
jr   $31
add  $29  $31   $0
addi $3  $0   14
addi $1  $0   0000
addi $2  $0   0001
addi $4  $0   0000
sw   $1  $4   0000
addi $4  $4   0004
sw   $2  $4   0000
add  $30  $1   $0
jal   8
```

```
00000000000000000000000000000000
001000_00000_111100000100100110010
000011_00000_000000000000000001000
000011_00000_000000000000000001010
000011_00000_000000000000000100011
001000_00000_111100000110101011101
000011_00000_000000000000000001000
000010_00000_000000000000000111010
101011_00000_1111000000_00100_000000
000000_11111_000000000000000001000
000000_11111_0000011101_00000_100000
001000_00000_0001100000_00000_001110
001000_00000_0000100000_00000_000000
001000_00000_0001000000_00000_000001
001000_00000_0010000000_00000_000000
101011_00100_0000100000_00000_000000
001000_00100_0010000000_00000_000100
101011_00100_0001000000_00000_000000
000000_00001_0000011110_00000_100000
000011_00000_000000000000000001000
000000_00010_0000011110_00000_100000
000011_00000_000000000000000001000
```

MIPS Converter

- https://goo.gl/CWkyQP
  - Input format
    - add t1 t2 t3
  - one line at a time



MIPS Converter

Instruction ⇒ Hex

add t1 t2 t3

ex. add t1 t2 t3, addi t1 t2 0xffff, j 0x02ffff

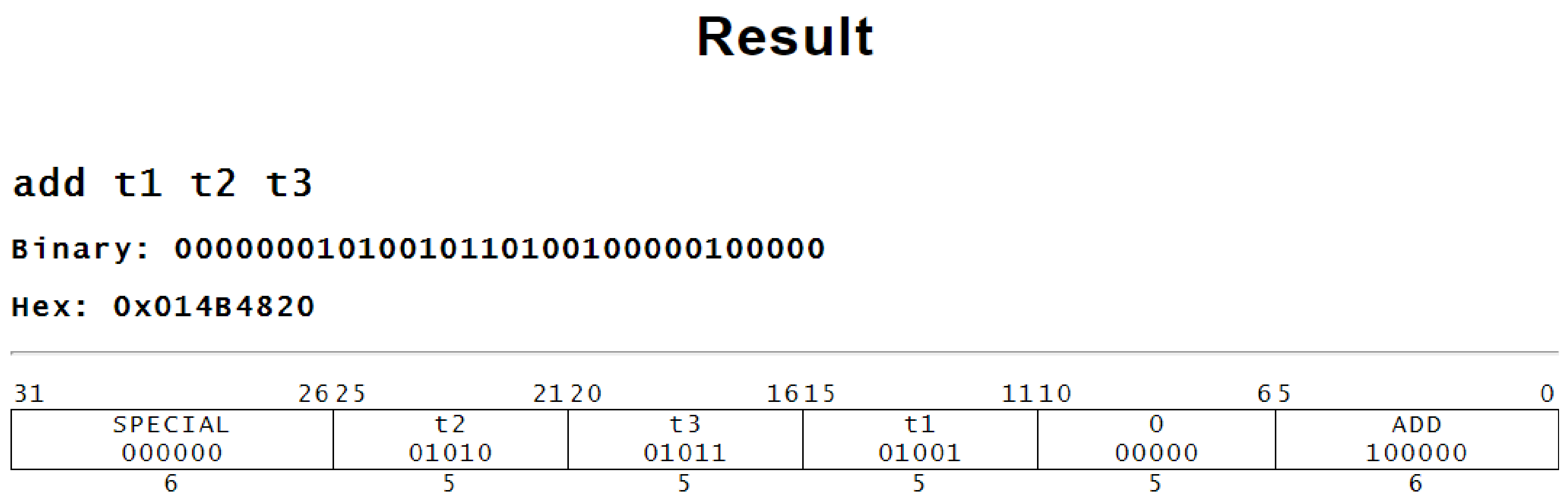
Convert

Hex ⇒ Instruction

Hex

ex. 0x014B4820

Convert



Result

```
add t1 t2 t3

Binary: 00000001010010110100100000100000

Hex: 0x014B4820
```

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 0
SPECIAL 000000	t2 01010	t3 01011	t1 01001	0 00000	ADD 100000
6	5	5	5	5	6

ADD

Add Word

Format:
ADD rd, rs, rt [R-type]

MIPS Architecture Extension: MIPS I

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	ADD 100000
6	5	5	5	5	6

Purpose:
To add 32-bit integers. If overflow occurs, then trap.

Description:
rd <- rs + rt
The 32-bit word value in GPR rt is added to the 32-bit value in GPR rs to produce a 32-bit result. If the addition results in 32-bit 2's complement arithmetic overflow then the destination register is not modified and an Integer Overflow exception occurs. If it does not overflow, the 32-bit result is placed into GPR rd.

Restrictions:
On 64-bit processors, if either GPR rt or GPR rs do not contain sign-extended 32-bit values (bits 63..31 equal), then the result of the operation is undefined.

Operation:
```
if (NotWordValue(GPR[rs]) or NotWordValue(GPR[rt])) then UndefinedResult() endif
temp <- GPR[rs] + GPR[rt]
if (32_bit_arithmetic_overflow) then
  SignalException(IntegerOverflow)
else
  GPR[rd] <- sign_extend(temp31..0)
endif
```

Exceptions:
Integer Overflow

Programming Notes:
ADDU performs the same arithmetic operation but, does not trap on overflow.

Implementation Notes: