h831 Assembly Programming Guide

Guide for programming h831 CPU in assembly langauge

Overview

This guide will cover the hardware features and serve as a reference for programming the H831 CPU using the assembly language.

Assembly Programming style for h831

General syntax

The general syntax for H831 assembly follows the below pattern.

Some instructions take in 3 arguments, while some take 2, 1 or even none. The section will specify the arguments for each instruction one-by-one.

Prefixes

The assembly langauge of h831 makes extensive use of prefixes to extinguish different objects (such as registers, literals and addresses). When calling (specifying) these objects, you must use an appropritate prefix.

Object	Prefix
General purpose registers	`%`
Internal registers	`_`
RAM Addresses/ IO Addresses	`$`
Literals (Immediate values)	`$`
Code offsets (Labels)	`.`
Comments (Ignored by assembler)	`;`

Specifying General Purpose Registers (GPR)

The h831 comes with 8 8-bit-general-purpose registers. A list of such registers and their preferred uses are listed below.

Register name	Assembly	Preferred purpose
x0	`%x0`	Accmulator, Storing return values
x1	`%x1`	No special purposes
x2	`%x2`	No special purposes
x3	`%x3`	No special purposes
x4	`%x4`	No special purposes
x5	`%x5`	No special purposes
x6	`%x6`	No special purposes
x7	`%x7`	Base pointer for stack (segmentation)

> **Special note**
>
> `%` is defined to be a prefix for specifying the register names in the assembly.
> If you would like to operate on registers, you must add the `%` in front of their names.
>
> E.g. `mov %x0, %x1` is valid, while `mov x0, x1` is invalid.

Specifying immediate values

Immediate values are the values that are hard-coded in the program code. There are 2 buses for immediate values, via A bus and B bus in the ALU. However, since B bus has some dedicated uses, the assembler will only use A bus for transferring immediate values.

The general syntax for specifying immediate values are as follows.

A prefix, namely `$`, is required to be placed in front of the decimal of the immediate value.

Example

To specify a 87, you would do `$87`.

Specifying code offset (Program address in branching)

In h831 assembly, you do not need to specify a fixed code offset when branching. You can use labels to mark the offset to be jumped to. However, do pay attention to the following remarks.

Concept of labels

Labels is just an lexical symbol to mark the point where a new section of program (sometimes called **subroutines**) begins. The use of lexical symbols (often an english word) for labels may act as a **heading** for a section of code.

For an example,

```
.check: pop %x2          ;       Read from stack                 6
        cmp %x0, %x2     ;       If memory write incorrect       7
        bnz .halt        ;       halt                            8
        dec %x0, %x0     ;       Decrement x0                    9
        cmp %x0, $1      ;       Do 32 times                     10
        bnz .check       ;                                       11
        not %x4,%x4      ;                                       12
```

In the above, `.check` is the label, which checks the RAM content byte by byte in some program.

Creating Labels

To create a label, add `.`labelName`:` to the beginning of the line that is intented to be marked. Note that the dot `.` is the prefix for labels. (To let the computer distinguish whether it is instruction or a label), and the `:` denotes the beginning of a new section.

Example

In the following assembly, line 1 is intented to be marked with a label named "loop".

```
        push %x0         ;       Write x0 to stack               3
        inc %x0          ;       Increment x0 by 1               4
        cmp %x0, %x1     ;       Compare x0 , x1                 5
        bnz .loop        ;
```

To mark it, add `.loop:` in the beginning of line 1.

```
1  .loop:  push %x0        ;       Write x0 to stack               3
2          inc %x0         ;       Increment x0 by 1               4
3          cmp %x0, %x1    ;       Compare x0 , x1                 5
4          bnz ?           ;       Jump reference not specified yet
```

Jumping to a specific code offset (using Labels)

To jump to a specific code offset, specifying its label right after the branching instruction.

Example

To jump back to `push %x0`, do `bnz .loop` in line 4.

```
1  .loop:  push %x0        ;       Write x0 to stack               3
2          inc %x0         ;       Increment x0 by 1               4
3          cmp %x0, %x1    ;       Compare x0 , x1                 5
4          bnz .loop       ;
```

Internal registers

Internal registers prohibit direct writing. The symbol of `_` is the prefix for internal registers.

CPSR (`_CPSR`)

CPSR stores the flags which describes the computational result from the ALU.

There are 3 flags available on this CPU. These are zero flag, carry flag and sign flag. The register `_CPSR` stores these flags.

CPSR stands for Current processor status register.

> **Future development**
>
> In later revisions there will be a **trap** flag, which halts the clock if it is called by **breakpt** instruction.

Stack Index (`_SI`)

Stack index stores the index where the stack have grown to.

Instructions

As of 2020-09-02, h831 supports a number of instructions. A table for them are below.

Instruction mnemonic	Meaning	No. of arguments take
mov	Copies data between registers	2

movi	Copies immediate value to register	2
add	Add the value in 2 registers and store result in anoter one	3
addi	Add the value in a register with an immediate value and store result in another register	3
sub	Subtract the value in 2 registers and store result in anoter one	3
subi	Subtract the value in a register with an immediate value and store result in another register	3
tcp	Compute two's complement for the value in a register, and store it in another register	2
and	Perform bitwise AND between 2 registers, and put result in a separate register	3
andi	Perform bitwise AND between an register and an immediate value, and put result in a separate register	3
xor	Perform bitwise XOR between 2 registers, and put result in a separate register	3
xori	Perform bitwise XOR between an register and an immediate value, and put result in a separate register	3
not	Perform bitwise NOT in an register, and store result in a separate register	2
inc	Increment the value in a register, and put the result into a separate register	2
dec	Decrement the value in a register, and put the result into a separate register	2
cmp	Compare the values in 2 registers and sets the status register (Flags)	2
cmpi	Compare the values in a registers with an immediate value and sets the status register (Flags)	
b	Unconditional branch	1
bz	Branch if zero flag is True	1
bnz	Branch if zero flag is False	1
bs	Branch if sign flag is True	1
bns	Branch if sign flag is False	1

bc	Branch if carry flag is True	1
bnc	Branch if carry flag is False	1
str	Store a value from a register to an address as immediate value in RAM	2
ldr	Load a value from the RAM with address as immediate value to a register	2
push	Pushes a value from a register into the stack, requires a desired `%x7` as base pointer	1
pop	Pops a value from the stack to a register, requires a desired `%x7` as base pointer	1
stkz	Sets stack pointer (`%SI`) to be 0	0
in	Read a value from an IO device to a register	2
out	Write a value from a register to an IO device	2
breakpt	Halts the clock	1
dbgout	Displays the value in data bus if attatched a debug 7-seg display	1

A detailed description about each instruction are below.

mov

Usage

Copies data between 2 registers.

Arguments

Takes in 2 arguments.

- `dest` : Register name, The register name to copy to
- `src` : Register name, The register name to copy from

Syntax

mov dest, src;

Example

```
mov %x0, %2;
```

movi

Usage

Copies immediate value to a register.

Arguments

Takes in 2 arguments.

- `dest`: Register name, The register name to copy to
- `val`: Literal, The immediate value to be copied

Syntax

movi dest, val;

Example

```
mov %x7, $87;
```

add

Usage

Add the value in 2 registers and store result in anoter one.

Arguments

Takes in 3 arguments.

- `dest`: Register name, The register name to store the result
- `src0`: Register name, The first value from the register to be added
- `src1`: Register name, The second value from the register to be added

Syntax

add dest, src0, src1;

Example

```
add %x4, %x7, %x8;
```

addi

Usage

Add the value in a register with an immediate value and store result in another register.

Arguments

Takes in 3 arguments.

- `dest`: Register name, The register name to store the result
- `src`: Register name, The source value from the register to be added
- `val`: Literal, The immediate value to be added

Syntax

addi dest, src, val;

Example

```
addi %x0, %x0, $89;
```

sub

Usage

Subtract the value in 2 registers and store result in anoter one.

Arguments

Takes in 3 arguments.

- `dest`: Register name, The register name to store the result
- `src0`: Register name, The first value from the register to subtract
- `src1`: Register name, The second value from the register to be subtracted

Remarks

Notice subtraction of any number, say, a, b, c if $c = a - b$ can be expressed as $c = a + (-b)$.
In this instruction, **`src0` would be the negative number that is being "added"**.

Syntax

sub dest, src0, src1

Example

```
sub %x0, %x5, %x0;
```

subi

Usage

Subtract the value in a register with an immediate value and store result in another register.

Arguments

Takes in 3 arguments.

- `dest`: Register name, The register name to store the result
- `src`: Register name, The first value from the register to be subtracted
- `val`: Literal, The immediate value to subtract

Remarks

Notice subtraction of any number, say, a, b, c if $c = a - b$ can be expressed as $c = a + (-b)$.
In this instruction, **`src` would be the negative number that is being "added"**.

Syntax

subi dest, src, val

Example

```
subi %x0, %x5, $4;
```

tcp

Usage

Compute two's complement for the value in a register, and store it in another register.

Arguments

Takes in 2 arguments.

- `dest`: Register name, The register name to store the result
- `src`: Register name, The source register containing the number to compute the two's complement

Syntax

tcp dest, src

Example

```
tcp %x5, %x5;
```

and

Usage

Perform bitwise AND between 2 registers, and put result in a separate register

Arguments

Takes in 3 arguments.

- `dest`: Register name, The register name to store the result
- `src0`: Register name, The first register containing the number to compute bitwise AND
- `src1`: Register name, The second register containing the number to compute bitwise AND

Syntax

and dest, src0, src1

Example

```
and %x0, %x0, %x1;
```

andi

Usage

Perform bitwise AND between a register and an immediate value, and store result in a separate register.

Arguments

Takes in 3 arguments.

- `dest`: Register name, The register name to store the result
- `src`: Register name, The source register containing the number to compute bitwise AND
- `val`: Literal, The immediate value to compute bitwise AND

Syntax

andi dest, src, val

Example

```
andi %x0, %x0, $5;
```

xor

Usage

Perform bitwise XOR between 2 registers, and put result in a separate register

Arguments

Takes in 3 arguments.

- `dest`: Register name, The register name to store the result
- `src0`: Register name, The first register containing the number to compute bitwise XOR
- `src1`: Register name, The second register containing the number to compute bitwise XOR

Syntax

xor dest, src0, src1

Example

```
xor %x5, %x0, %x1;
```

xori

Usage

Perform bitwise XOR between a register and an immediate value, and store result in a separate register.

Arguments

Takes in 3 arguments.

- `dest`: Register name, The register name to store the result
- `src`: Register name, The source register containing the number to compute bitwise XOR
- `val`: Literal, The immediate value to compute bitwise XOR

Syntax

xori dest, src, val

Example

```
xori %x0, %x7, $5;
```

not

Usage

Perform bitwise NOT of the value in a registe and store result in a separate register.

Arguments

Takes in 2 arguments.

- `dest`: Register name, The register name to store the result
- `src`: Register name, The source register containing the number to compute bitwise NOT

Syntax

not dest, src

Example

```
not %x0, %x7;
```

inc

Usage

Increment the value in a register, and put the result into a separate register.

Arguments

Takes in 2 arguments.

- `dest`: Register name, The register name to store the result
- `src`: Register name, The source register containing the number to increment

Syntax

inc dest, src

Example

```
inc %x2, %x7;
```

dec

Usage

Decrement the value in a register, and put the result into a separate register.

Arguments

Takes in 2 arguments.

- `dest`: Register name, The register name to store the result
- `src`: Register name, The source register containing the number to decrement

Syntax

dec dest, src

Example

```
dec %x5, %x4;
```

cmp

Usage

Compare the values in 2 registers and sets the status register (Flags).

Arguments

Takes in 2 arguments.

- `src0`: Register name, The value in the first register to be compared
- `src1`: Register name, The value in the second register to be compared

Syntax

cmp src0, src1

Remarks

There are 3 flags available on this CPU. These are zero flag, carry flag and sign flag. The register `_CPSR` stores these flags.

CPSR stands for Current processor status register. The `_` is the prefix for internal registers.

Example

```
cmp %x5, %x4;
```

cmpi

Usage

Compare the values between a value in the register and an immediate value. Sets the status register (Flags).

Arguments

Takes in 2 arguments.

- `src`: Register name, The value in the register to be compared
- `val`: Literal, The immediate value to be compared

Syntax

cmpi src, val

Remarks

There are 3 flags available on this CPU. These are zero flag, carry flag and sign flag. The register `_CPSR` stores these flags.

CPSR stands for Current processor status register. The `_` is the prefix for internal registers.

Example

```
cmpi %x5, $64;
```

b

Usage

Unconditional branch.

Arguments

Takes in 1 argument.

- `offset`: Code offset, The label of marking the code offset to be jumped.

Syntax

b offset

Example

```
b .main;    Provided that .main is declared above
```

bz

Usage

Branch if zero flag is True.

Arguments

Takes in 1 argument.

- `offset`: Code offset, The label of marking the code offset to be jumped.

Syntax

bz offset

Example

```
bz .loop;   Provided that .loop is declared above
```

bnz

Usage

Branch if zero flag is False.

Arguments

Takes in 1 argument.

- `offset`: Code offset, The label of marking the code offset to be jumped.

Syntax

bnz offset

Example

```
bnz .loop;  Provided that .loop is declared above
```

bs

Usage

Branch if sign flag is True.

Arguments

Takes in 1 argument.

- `offset`: Code offset, The label of marking the code offset to be jumped.

Syntax

bs offset

Example

```
bs .redo;   Provided that .redo is declared above
```

bns

Usage

Branch if sign flag is False.

Arguments

Takes in 1 argument.

- `offset`: Code offset, The label of marking the code offset to be jumped.

Syntax

bns offset

Example

```
bns .redo;  Provided that .redo is declared above
```

bc

Usage

Branch if carry flag is True.

Arguments

Takes in 1 argument.

- `offset`: Code offset, The label of marking the code offset to be jumped.

Syntax

bc offset

Example

```
bc .recheck;    Provided that .recheck is declared above
```

bnc

Usage

Branch if carry flag is False.

Arguments

Takes in 1 argument.

- `offset`: Code offset, The label of marking the code offset to be jumped.

Syntax

bnc offset

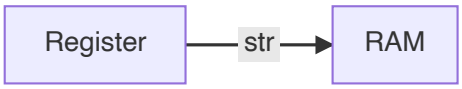
Example

```
bnc .recheck;   Provided that .recheck is declared above
```

str

Usage

Store a value from a register to an address in RAM.



Arguments

Takes in 2 arguments.

- `dest`: RAM Address, The address pointing to a location in RAM to be read
- `addr`: Register name, The value read from RAM is stored in this register.

Syntax

str dest, addr

Example

```
1 str $23, %x5;
```

ldr

Usage

Load a value from an address in RAM to a register.

Arguments

Takes in 2 arguments.

- `dest`: Register name, The value read from RAM is stored in this register.
- `addr`: RAM Address, The address pointing to a location in RAM to be read

Syntax

ldr dest, addr

Example

```
1 ldr %x5, $23
```

push

Usage

Pushes a value from a register into the stack, requires a desired `%x7` as base pointer.

Arguments

Takes in 1 argument.

- `src`: Register name, The value in the register to be pushed.

Syntax

push src

Example

```
push %x5;
```

pop

Usage

Pops a value from a register into the stack, requires a desired `%x7` as base pointer.

Arguments

Takes in 1 argument.

- `dest`: Register name, The register to store the popped value.

Syntax

pop dest

Example

```
pop %x0;
```

stkz

Usage

Sets stack pointer (`_SI`) to be 0. Abbreviated from "**Stack Zero**".

Arguments

Takes no arguments

Syntax

stkz

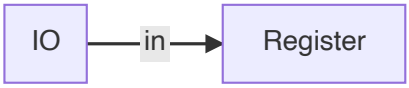
Example

```
stkz;
```

in

Usage

Reads a value from an IO device to a register.



Arguments

Takes in 2 arguments.

- `dest`: Register name, The register to store the value read from the IO device.
- `src`: IO Address, The IO device containg the value to be sent.

Syntax

in dest, src

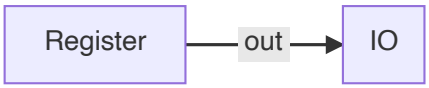
Example

```
in %x4, $1;
```

out

Usage

Outputs a value from an register to an IO device.



Arguments

Takes in 2 arguments.

- `dest`: IO Address, The IO device to store the value read from the register.
- `src`: Register name, The register restoring the value to be sent

Syntax

out dest, addr

Example

```
out $5, %x1;
```

breakpt

Usage

Halts the clock unconditonally. Shorten from **Breakpoint**.

> **Future development**
>
> Halts the clock if the **trap** flag is set.

Arguments

Takes no arguments.

Syntax

breakpt

Example

```
breakpt;
```

dbgout

Usage

Displays the value in data bus if attatched to a debug 7-seg display.

Arguments

Takes no arguments.

Syntax

dbgout

Example

```
dbgout;
```

Version information

1. 2020-09-03 Revision. Added description for essential commands.
    - Future development: Trap flag

> **Footnote**
>
> This is my first attempt on building a MC computer.
> There are design flaws everywhere, but I have a lot of fun building this.
> I would like to especially thanks to n00b_asaurus. I follow his great guide and tutorial videos on building this minecraft computer. I regret to watch it so late.