ASIP Meister User’s Manual

Revision 1.1.2
PEAS Project

December 13, 2004

Table of Contents

List of Figures

List of Tables

I. Introduction

1. About ASIP Meister Documents

Table 1 shows the documents list of ASIP Meister. The table describes the name of the document, its contents and its path from ASIP Meister directory. If you follow the “Installation Guide”, ASIP Meister directory is “/usr/local/ASIPmeister “. These related documents provide supplementary explanation of this manual. Please refer to them as necessary.

Table 1: ASIP Meister documents list

Title	Contents
ASIP Meister User’s manual (downloadable from ASIP Meister web page)	This manual
Install Guide (InstallGuide.txt)	It explains the required hardware and software to run ASIP Meister how to install and setup ASIP Meister and how to uninstall ASIP Meister.
ASIP Meister Tutorial (downloadable from ASIP Meister web page)	It is the tutorial document. It helps you to familiarize yourself with ASIP Meister.
Processor DLX Integer Specification (English: share/tutorial/dlx_integer/dlx_integer.pdf Japanese: share/tutorial/dlx_integer/dlx_integer_jpn.pdf)	Specification of the sample data
Micro-Operation Description Coding Guide (share/microOpe_0.5.pdf)	It is coding guideline of micro-operation description.

Furthermore, we provide the bibliography list of the research paper at the end of this manual for your further reference.

2. Manual Convention

This manual targets for those who are beginners to use ASIP Meister and those who would like to confirm the specific functionality of ASIP Meister.

This manual has the following conventions;

- User interfaces such as menus, buttons, text boxes and checkboxes are enclosed with the brackets, like [Save Design As...].
- “%” symbol means the command prompt.
- The name of files and directories are enclosed with the double quotation marks, like “/usr/local/ASIPmeister.

II. How to use ASIP Meister

1. Basic operation

This chapter explains the basic operation of ASIP Meister.

1.1. Start ASIP Meister

ASIP Meister requires Java™2 environment. Please install Java™2 first. The rest of this section assumes that the Java™2 has been correctly installed to your computer.

When you start ASIP Meister, the path to ASIP Meister and the Java™2 running environment needs to be specified. The sample file (“ASIPmeister.setup.sample”) is provided at the “/usr/local/ASIPmeister/share” directory. This sample setup file assumes that your Java™2 running environment is installed at the “/usr/java/jre1.3.1_04” directory. If it is installed in a different directory, please modify the settings according to your environment. This sample file is also written for bsh shell, so if you are using csh or tsch, please change your settings as below;

```
setenv PATH =/usr/local/j2sdk1.3.1/bin/:$PATH
setenv PATH =/usr/local/ASIPmeister/bin/:$PATH
```

The ASIP Meister start command is “ASIPmeister“. As in the following example, you can specify the input data file as an argument.

```
%   ASIPmeister [input_data_file.pdb]
```

”.pdb” is the extension of the data file of ASIP Meister.

1.2. The main menu window

When you start ASIP Meister, [ASIP Meister Main Menu] window (Figure 1) opens.

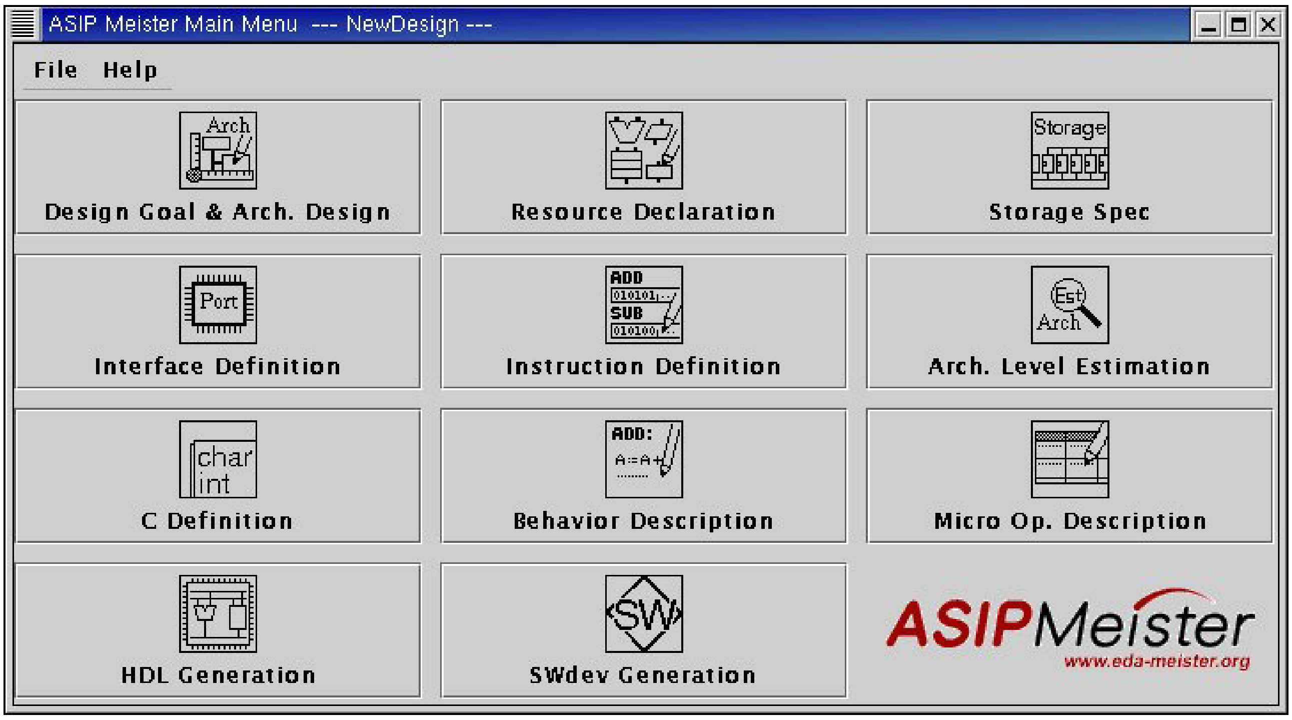


Figure 1: [ASIP Meister Main Menu] window

The title bar of the main menu window displays the data file currently under design. For example, when you create a new design, "---New Data---" is displayed on the title bar.
The details of menus of [ASIP Meister Main Menu] window are described next.

1.2.1. [File] menu

Figure 2 shows [File] menu in [ASIP Meister Main menu].

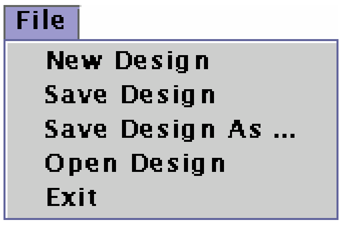


Figure 2: [File] menu of [ASIP Meister Main Menu] window

- **[New Design]**

This menu command starts a new design.

- **[Save Design]**

This menu command saves the design with the same name.

- **[Save Design As …]**

This menu command saves the design under a different name.
If you click [Save Design As...] menu, [Save] window opens (Figure 3). First, click [Look in] pull down menu to select the directory to save the data file. Then, enter the data file name to [File name] field and click [Save]

button to save the data file. The extension ".pdb" is automatically added to the data file name. You must not use reserved words described in appendix section 4.1 as the data file name.

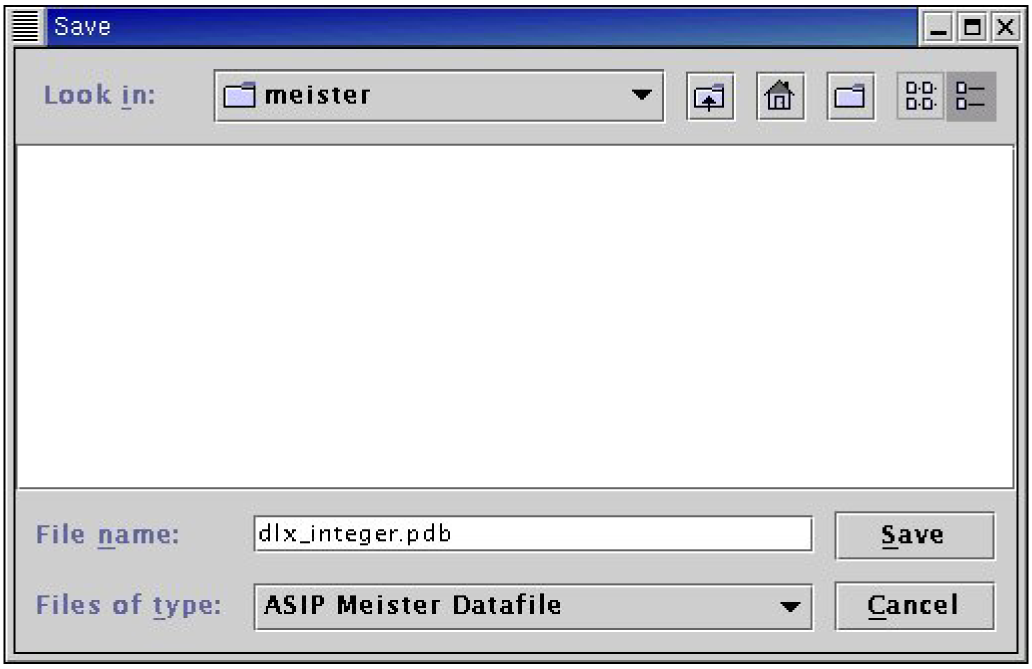


Figure 3: [Save] window

- **[Open Design]**

  This menu command loads the saved design data.
  If you click this menu command, the window to select the input data file opens. Select the data file to enter and click [Open] button.

- **[Exit]**

  This menu command exits ASIP Meister.

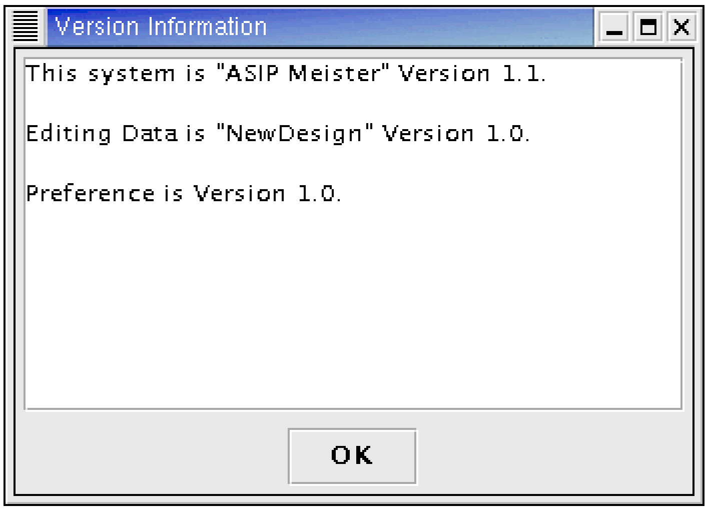
1.2.2. [Help] menu

Figure **4** shows [Help] menu in [ASIP Meister Main menu].

Figure 4: [Help] menu of [ASIP Meister Main Menu] window

- **[Version]**

  This menu command displays the version information of ASIP Meister (Figure 5).



This system is "ASIP Meister" Version 1.1.

Editing Data is "NewDesign" Version 1.0.

Preference is Version 1.0.

Figure 5: [Version Information] window

1.3. Sub windows

To open the sub window of each item in [ASIP Meister Main Menu] window, simply click the item in the window. The detail of how to set and enter the value for each flow is explained in the next chapter.

To close the sub window, click [File] -> [Close].

If you have completed to set all the options in the sub window, click [Complete] checkbox (Figure 6) before closing it. Checking [Complete] checkbox implies you have completed setting the options in the sub window. Then you can proceed and set options in another sub window. If you close the sub window without checking [Complete] checkbox, you cannot proceed to next sub window.

If a sub window is opened when you open other sub window, the title of the new sub window shows "READ ONLY" and it does not allow you to change the settings. When you proceed to the next window setting, please close the current sub window first.

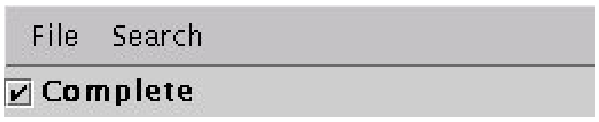


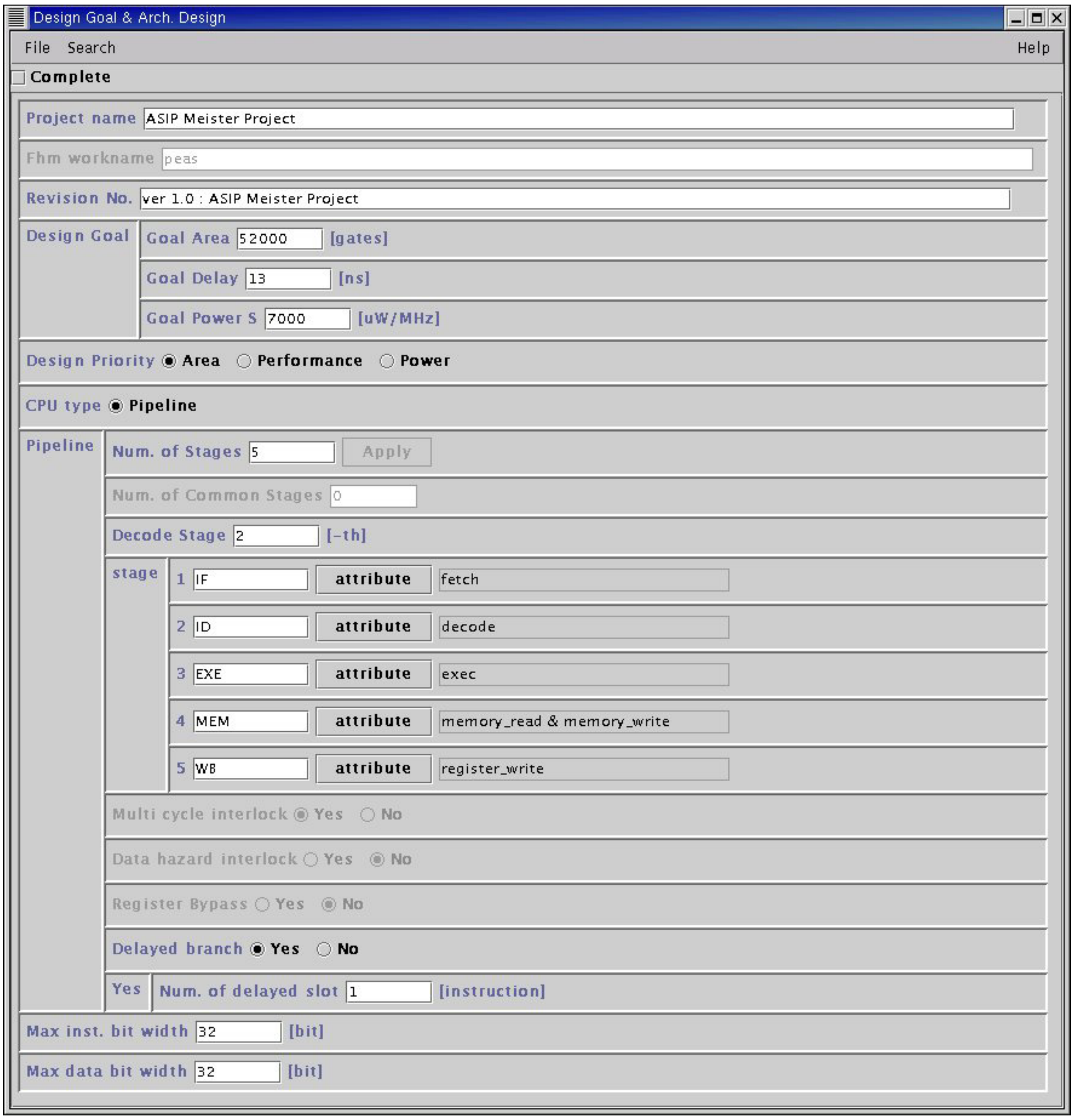
Figure 6: [Complete] checkbox of sub window

1.4. Working directory

ASIP Meister stores its working files and auto generated HDL description files to the "meister" directory. This working directory exists in the current directory. If it does not exist, ASIP Meister makes the directory automatically.

2. Design Goal & Arch. Design

[Design Goal &Arch. Design] window (Figure 7) opens by clicking [Design Goal. &Arch.] item in [ASIP Meister Main Menu] window. With this sub window, you can set the processor type, architecture parameters and processor specification.



Complete

Project name	ASIP Meister Project
Fhm workname	peas
Revision No.	ver 1.0 : ASIP Meister Project
Design Goal – Goal Area	52000 [gates]
Design Goal – Goal Delay	13 [ns]
Design Goal – Goal Power S	7000 [uW/MHz]
Design Priority	[✓ Area] Performance Power
CPU type	[✓ Pipeline]
Pipeline – Num. of Stages	5
Pipeline – Num. of Common Stages	0
Pipeline – Decode Stage	2 [-th]
stage 1	IF – fetch
stage 2	ID – decode
stage 3	EXE – exec
stage 4	MEM – memory_read & memory_write
stage 5	WB – register_write
Multi cycle interlock	[✓ Yes] No
Data hazard interlock	Yes [✓ No]
Register Bypass	Yes [✓ No]
Delayed branch	[✓ Yes] No
Yes – Num. of delayed slot	1 [instruction]
Max inst. bit width	32 [bit]
Max data bit width	32 [bit]

Figure 7: [Design Goal & Arch. Design] sub window

- **[Project name]: Project name**

Enter the project name for the current design data to this field. You can use any characters for this field. So use a project name that is most convenient for your design.

The project name entered to this field is not the entity name for the processor. You can specify the entity name for the processor in [Interface Definition] window.

- **[Fhm workname]: Resource data base name**

This field shows the database name of the FHM resource used by ASIP Meister.
In this version, you cannot modify the name.

- **[Revision No.]: Revision number**

Enter the version information of the design data to this field. You can use any characters for this field. Use this field to manage the version information of the design.

- **[Design Goal]: Target value of the design**

Enter the target design qualities of the processor to these fields. Enter the target area of the processor, the target delay of the data path (combination circuit) and the target static power consumption to [Goal Area], [Goal Delay] and [Goal Power S] field respectively. Architecture-level estimation engine uses these values (for details refer to section 7).

- **[Design Priority]: Priority of the design target**

Select which option ([Area], [Performance] or [Power]) to prioritize in the design target of the processor. If you click [Area] radio button, the area of the processor will have the highest priority when architecture-level estimation engine estimate the design qualities. Likewise, if you click either [Performance] or [Power] radio button, the priority of the estimation changes accordingly.

- **[CPU type]: Processor type to design**

This field shows the architecture type of the processor to design.
In this version, ASIP Meister only supports pipeline architecture.

- **[Num. of Stages]: The number of the pipelines**

Enter the number of the pipeline stages to this field. Click [Apply] button, and then the number of the rows that appear in [stage] field corresponds to the value entered in [Num. of Stages] field. The blank rows are added when you increase the number of stages. When you decrease the number of stages, the row are deleted from the last row.
If you decrease the number of stages, [Change stage number Confirm] window will open when you select [File] > [Close] menu. In [Micro Op. Description] phase, you must describe the behavior of instruction par stage. So, you must select [Merge] for merging the behavior of the deleted stages or [Discard] for discarding the behavior of the deleted stages.

- **[Num. of Common Stages]: Number of common stages in the pipeline**

This field shows the number of common stages.
In this version, you cannot modify the value.

- **[Decode Stage]: Decoding stage**

Enter the stage number which stage executes the instruction decoding to this field.

- **[Stage]: Stage information**

These fields have rows to enter the parameter for each pipeline stage. The number of the row displayed corresponds to the value entered in [Num. of Stages] field. Each row displays the stage number, the stage name and its attributes. You must not use reserved words described in appendix section 4.1 as the stage name.
You can set the stage attribute with the attribute selection window (Figure 8). That window opens by clicking [attribute] button. Click the checkbox of the attribute to use for the stage and click [OK] button to close the window. The attributes setting appears in stage specification filed. You can set multiple attributes to each stage. You can set attributes shown in Table 2.

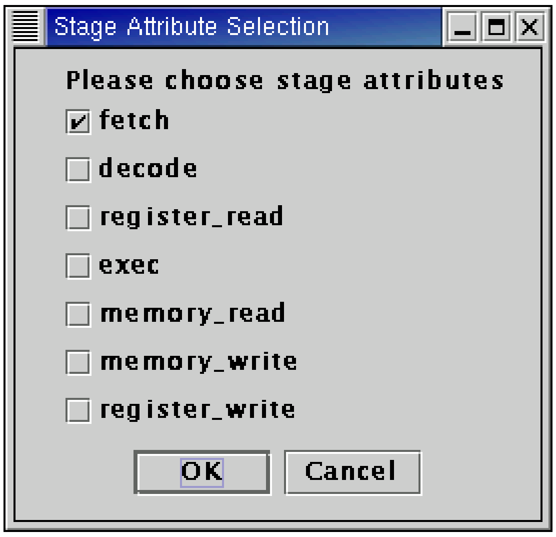


Figure 8: Attribute selection window for pipeline stage

Table 2: Attributes of the pipeline stage

Attribute	Description
[fetch]	Instruction fetch stage
[decode]	Instruction decoding stage
[register_read]	Register reading stage
[exec]	Operation execution stage
[memory_read]	Memory reading stage
[memory_write]	Memory writing stage
[register_write]	Register writing stage

- **[Multi cycle interlock]: Interlock setting for multi cycle operation**

This field shows the interlock setting for multi cycle operation.
In this version, you cannot modify the value.

- **[Data hazard interlock]: Interlock setting for data hazard**

This field shows the interlock setting for data hazard. In this version, you cannot modify the value.

- **[Register bypass]: Register bypass setting**

This field shows the register bypass (forward) setting. In this version, you cannot modify the value.

- **[Delay branch]: Delay branch setting**

Click [Yes] radio button, if you wish to use the delay branch.

- **[Num. of delayed slot]: Number of delay branch slots**

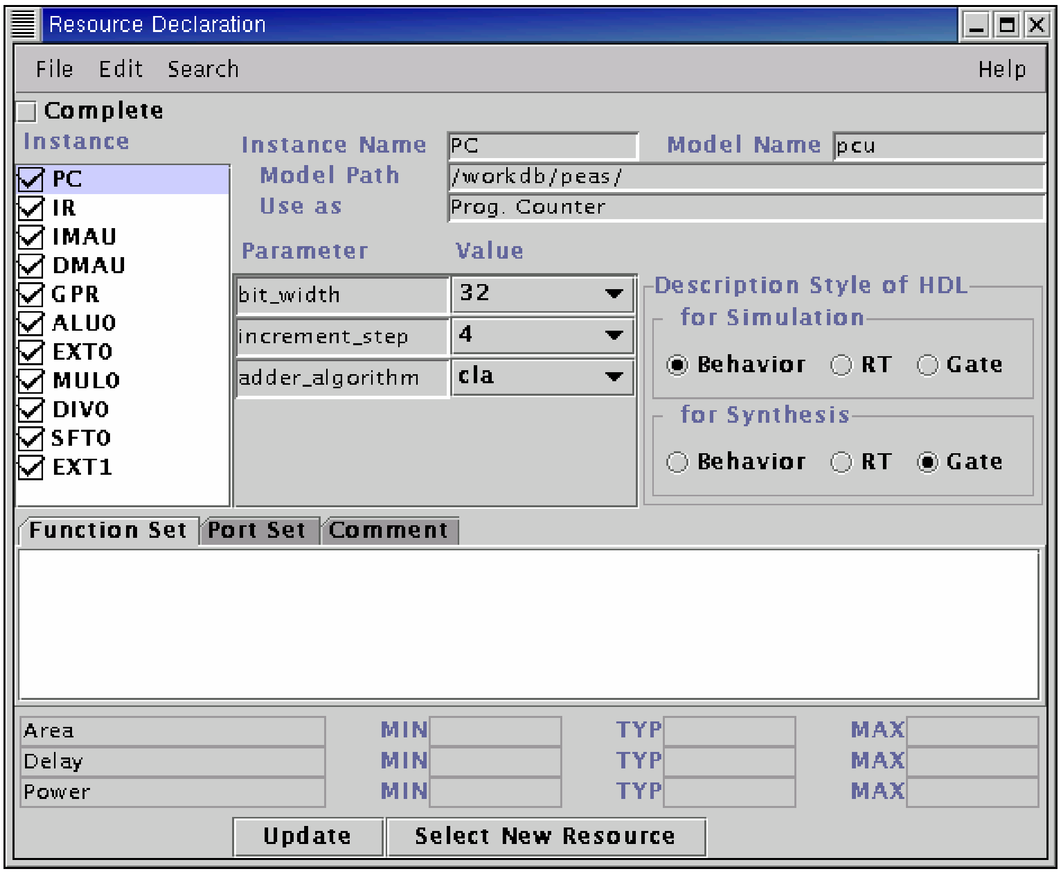
Enter the number of delayed slots to this field. This field becomes available only when you have selected [Yes] radio button at [Delayed Branch] field.

- **[Max inst. bit width], [Max data bit width]: Maximum bit width of the instruction and data**

Enter the maximum bit width of the instruction and data to these fields.

3. Resource Declaration

[Resource Declaration] window (Figure 9) opens by clicking [Resource Declaration] item in [ASIP Meister Main Menu] window. You can declare the resources of the processor in this window.



[ ] Complete

Instance: [✓] PC, [✓] IR, [✓] IMAU, [✓] DMAU, [✓] GPR, [✓] ALU0, [✓] EXT0, [✓] MUL0, [✓] DIV0, [✓] SFT0, [✓] EXT1

Instance Name	PC	Model Name	pcu
Model Path	/workdb/peas/		
Use as	Prog. Counter		

Parameter	Value
bit_width	32
increment_step	4
adder_algorithm	cla

Description Style of HDL
for Simulation: [✓ Behavior] RT Gate
for Synthesis: Behavior RT [✓ Gate]

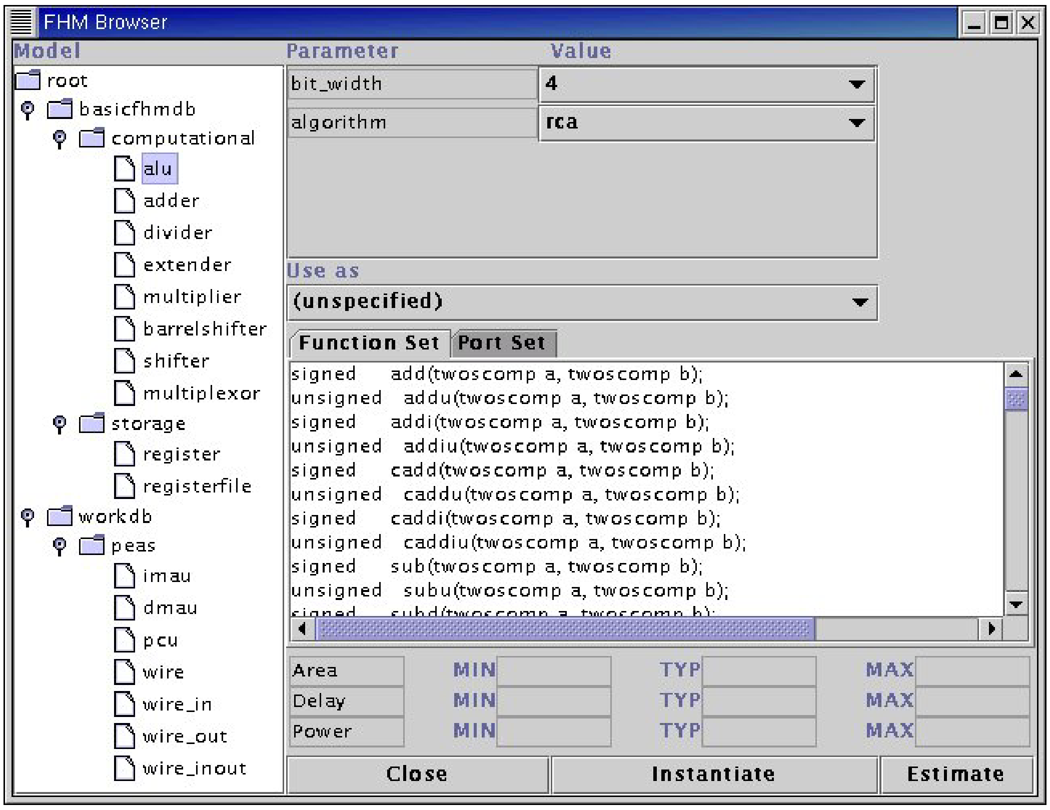
Area	MIN	TYP	MAX
Delay	MIN	TYP	MAX
Power	MIN	TYP	MAX

Figure 9: [Resource Declaration] window

You can select resources to implement from Flexible Hardware Model Database (FHM-DB).

3.1. FHM browser

[FHM Browser] window (Figure 10) lists all the registered resources. You can create the resource instance by selecting the resource and setting its parameters.



```
root
  basicfhmdb
    computational
      alu
      adder
      divider
      extender
      multiplier
      barrelshifter
      shifter
      multiplexor
    storage
      register
      registerfile
  workdb
    peas
      imau
      dmau
      pcu
      wire
      wire_in
      wire_out
      wire_inout
```

Parameter	Value
bit_width	4
algorithm	rca

Use as: (unspecified)

```
signed     add(twoscomp a, twoscomp b);
unsigned   addu(twoscomp a, twoscomp b);
signed     addi(twoscomp a, twoscomp b);
unsigned   addiu(twoscomp a, twoscomp b);
signed     cadd(twoscomp a, twoscomp b);
unsigned   caddu(twoscomp a, twoscomp b);
signed     caddi(twoscomp a, twoscomp b);
unsigned   caddiu(twoscomp a, twoscomp b);
signed     sub(twoscomp a, twoscomp b);
unsigned   subu(twoscomp a, twoscomp b);
signed     subd(twoscomp a, twoscomp b);
```

Area	MIN		TYP		MAX	
Delay	MIN		TYP		MAX	
Power	MIN		TYP		MAX	

Figure 10: FHM Browser

The following describes the options of [FHM Browser] window.

- **[Model]: Resource model list**

This field lists the available resources. If you click the resource in the list, [Parameter] field appears on the right side of [FHM Browser].

- **[Parameter], [Value]: Parameter item and its value of the resource**

[Parameter] field displays the available parameter options, according to the resource selected in [Model] field. Select the value from [Value] list for each parameter.

- **[Use as]: Usage of the resource**

Click the pull-down menu to select the role of the resource. For example, if you use register model as a program counter, select [Prog. Counter] from the list. Table 3 shows the options in the pull-down menu.

Table 3: Options of [Use as] pull-down menu

Option	Description
Inst. Register	Instruction register
Register File	Register file
Prog. Counter	Program counter
Inst. Memory	Instruction memory
Data Memory	Data memory
Plain Register	Plain register
Mask Register	Mask register (for interrupt)
(unspecified)	Select this option, when any other options in the list do not apply to your usage.

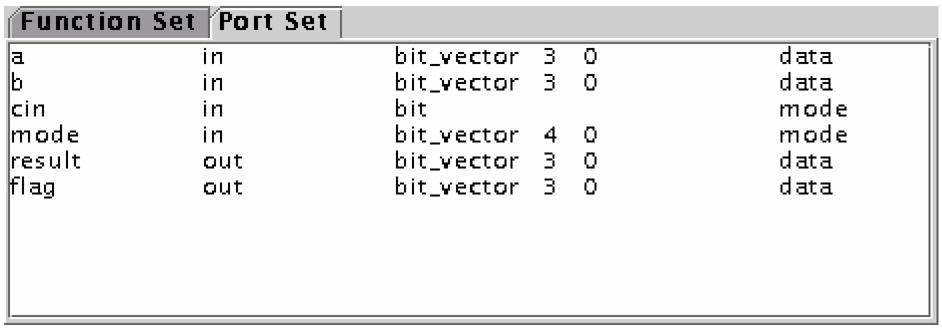
In ASIP Meister, each processor must include at least one resource as [Inst. Register], [Prog. Counter], [Inst. Memory] and [Data Memory].

- **[Function Set]: Function of the resource**

Click [Function Set] tab to check the function of the resource selected in the list on [Model] field. Use this option to check whether the selected resource model provides the intended function that you wish or not.

- **[Port Set]: I/O interface of the resource**

Click [Port Set] tab to view the port name, direction of I/O, data type, bit width and attribute of the signal (Figure 11).



```
a        in     bit_vector  3  0        data
b        in     bit_vector  3  0        data
cin      in     bit                     mode
mode     in     bit_vector  4  0        mode
result   out    bit_vector  3  0        data
flag     out    bit_vector  3  0        data
```

Figure 11: [Port Set] field

Direction of I/O is [in] for input port, [out] for output port or [inout] for input and output port. The bit width is described with VHDL description format. The Table 4 lists the attributes.

Table 4: Attribute list

Attribute	Description
clock	Clock signal
reset	Reset signal
ctrl	Resource control signal
data	Data signal
mode	Mode selection signal

- **[Estimate]: Estimation of resource**

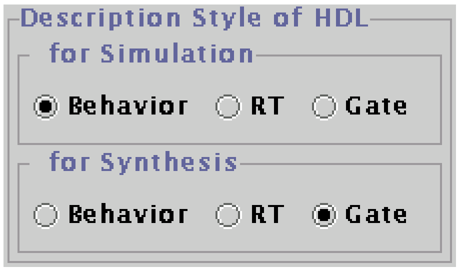
If you click [Estimate] button, the value for the area, delay and power consumption are estimated based on the parameters you have set. These estimated values are displayed in [Area], [Delay] and [Power] row respectively. Also each row displays three different estimated values. [MIN] field shows the minimum estimated value and [MAX] shows the maximum estimated value. [TYP] shows the estimated value with the design priority that you set in [Design Goal & Arch. Design] step.

- **[Instantiate]: Create resource instance**

Click [Instantiate] button, and then [New Instance] window opens. You can set the instance name in the window. Setting the instance name and clicking [OK] button, the instance will appear in [Resource Declaration] window. You must not use reserved words described in appendix section 4.1 as the instance name.

3.2. Abstraction level of the resource

You need to specify the abstraction level of both the simulation and the logical synthesis model. The level may be selected from [Behavior] for the functional level, [RT] for register transfer level and [Gate] for gate level (Figure 12).



Description Style of HDL

for Simulation

[✓ Behavior] RT Gate

for Synthesis

Behavior RT [✓ Gate]

Figure 12: Abstraction level of the resource

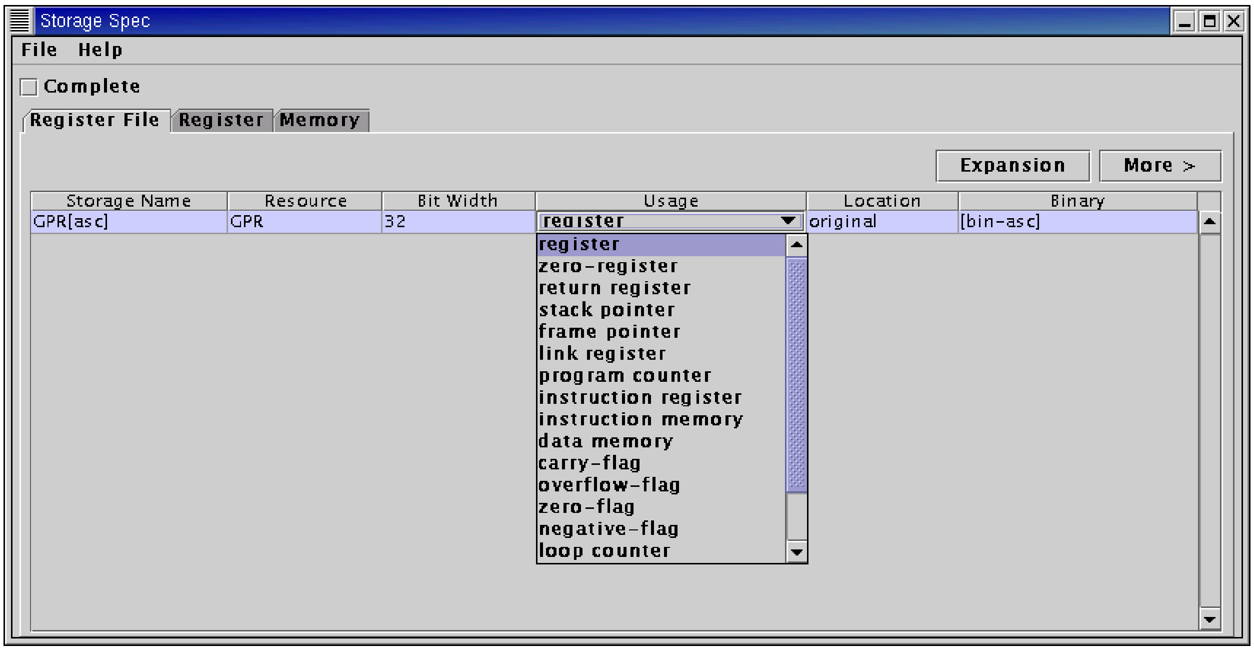
4. Storage Specification

In [Storage Spec] window, you need to set the specification of storages such as register files, registers and memories. The storage specification includes the name, data bit width and usage of the storage. Software tools generation engine will use the storage information.

When you click [Storage Spec.] button in [ASIP Meister Main Menu], you can find [Storage Spec] window. The storage specification consists of three parts; register file, register and memory.

4.1. Register file specification setting

When you select [Register File] tab in [Storage Spec] window, you can define the specification of the register files (see Figure 13). Table 5 shows the register file specification.



Storage Name	Resource	Bit Width	Usage	Location	Binary
GPR[asc]	GPR	32	register	original	[bin-asc]

Figure 13: Register file specification

Table 5: Specification of register file

Specification	Description
Storage name	Storage name used in assembly code
Resource	Resource name declared in [Resource Declaration] step
Bit Width	Storage data bit width
Usage	Storage usage that specify how to handle the storage in the compiler
Location	Storage location
Binary	Binary representation for machine code

Since register file has two or more registers, the number of registers can be specified using the following words;

- [asc]: ascending order
- [dsc]: descending order

Moreover, binary representation of each register can be specified using the following words;

- [binary-asc]: binary representation of ascending order
- [binary-dsc]: binary representation of descending order

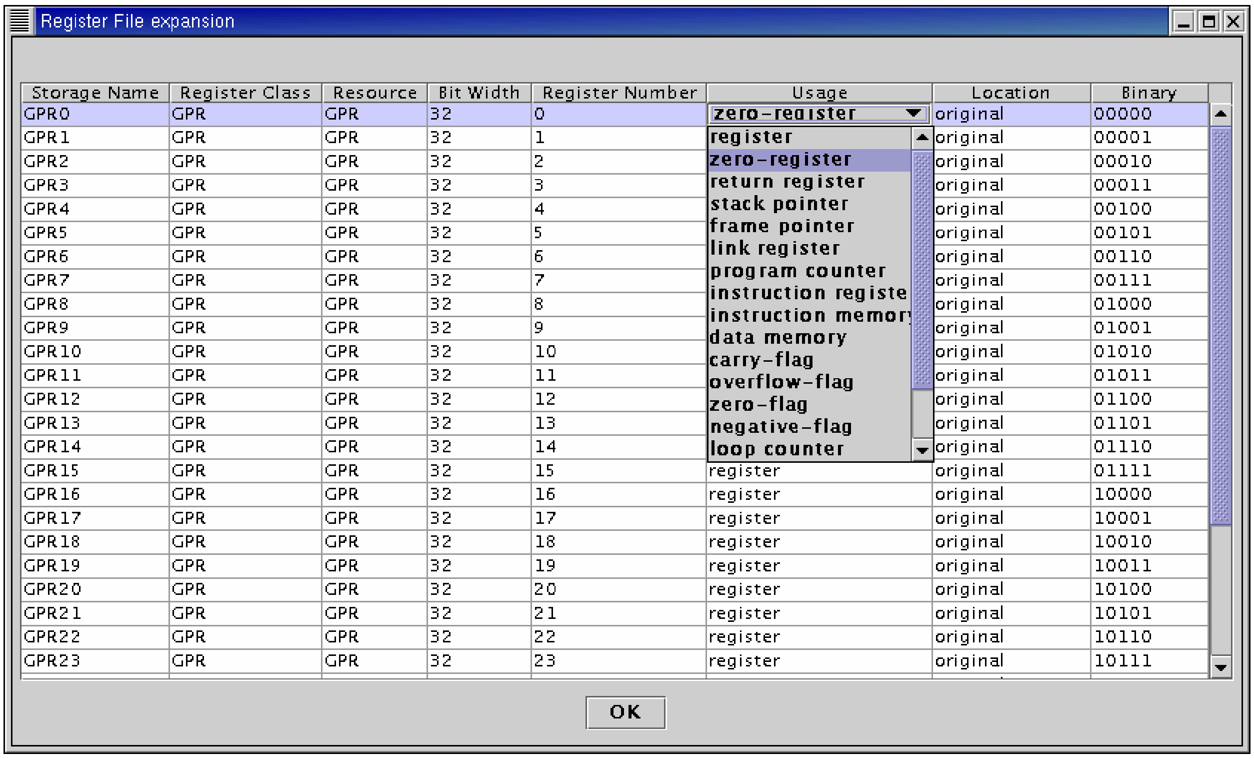
Storage usage can be selected from the list. Table 6 shows the storage usage. Compiler generation engine uses this information.

Table 6: Storage usage

Usage	Description
register	Data register
zero-register	Register with zero value
return register	Register that keeps the return value of the functional call
stack pointer	Register used as the stack pointer
frame pointer	Register used as the frame pointer
link register	Register that keeps the return address
program counter	Program counter that keeps the memory address for the instruction
instruction register	Register that keeps the instruction
instruction memory	Memory that contains instructions
data memory	Memory that contains data
carry-flag	Register that contains carry flag data
overflow-flag	Register that contains overflow flag data
zero-flag	Register that contains zero flag data
negative-flag	Register that contains negative flag data
loop counter	Loop counter for zero-overhead loop (future extension)
start-address register	Start address for zero-overhead loop (future extension)
end-address register	End address for zero-overhead loop (future extension)
instruction number	Register that keeps the number of instructions that specifies for zero-overhead loop block (future extension)

In the register file specification, just write "original" to [Location] field.

In register file definition, designers can specify the register specification to each register. When you click [Expansion] button, the window with the expanded storage list opens (Figure 14). Usage of each register and other features can be specified in this window. [More] button also shows the expanded lists but it does not reflect the change of the register file specification. If you want to check the previous setting, push [More] button, otherwise, push [Expansion] button.



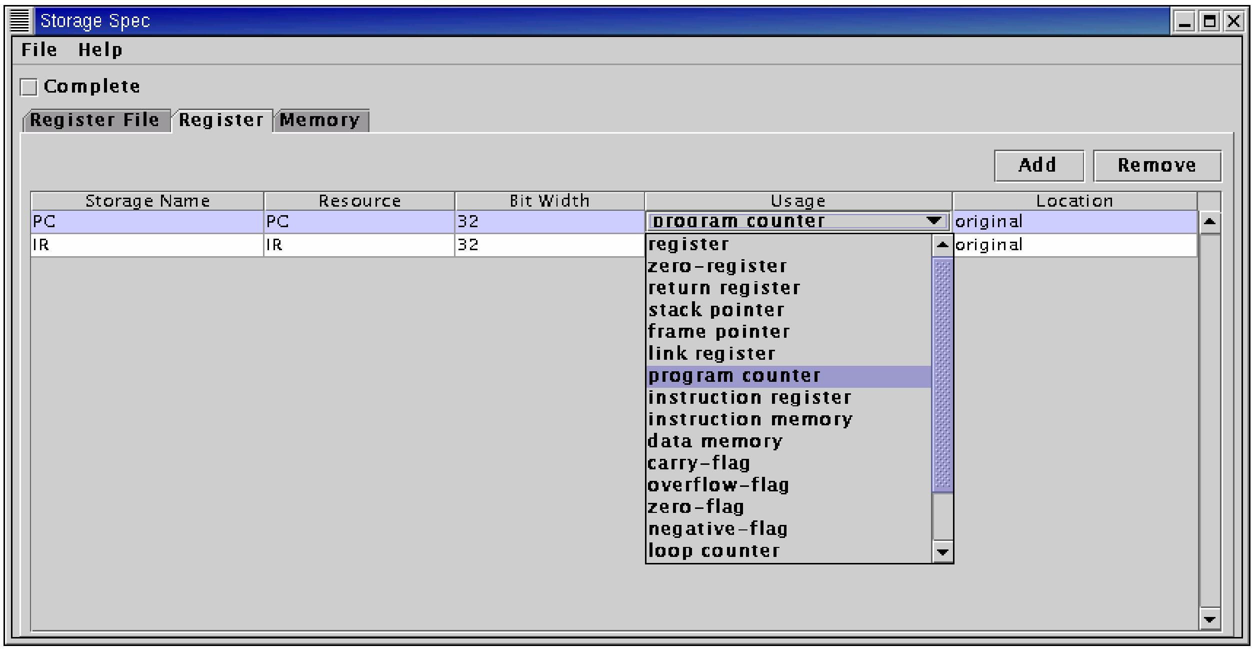
Storage Name	Register Class	Resource	Bit Width	Register Number	Usage	Location	Binary
GPR0	GPR	GPR	32	0	zero-register	original	00000
GPR1	GPR	GPR	32	1		original	00001
GPR2	GPR	GPR	32	2		original	00010
GPR3	GPR	GPR	32	3		original	00011
GPR4	GPR	GPR	32	4		original	00100
GPR5	GPR	GPR	32	5		original	00101
GPR6	GPR	GPR	32	6		original	00110
GPR7	GPR	GPR	32	7		original	00111
GPR8	GPR	GPR	32	8		original	01000
GPR9	GPR	GPR	32	9		original	01001
GPR10	GPR	GPR	32	10		original	01010
GPR11	GPR	GPR	32	11		original	01011
GPR12	GPR	GPR	32	12		original	01100
GPR13	GPR	GPR	32	13		original	01101
GPR14	GPR	GPR	32	14		original	01110
GPR15	GPR	GPR	32	15	register	original	01111
GPR16	GPR	GPR	32	16	register	original	10000
GPR17	GPR	GPR	32	17	register	original	10001
GPR18	GPR	GPR	32	18	register	original	10010
GPR19	GPR	GPR	32	19	register	original	10011
GPR20	GPR	GPR	32	20	register	original	10100
GPR21	GPR	GPR	32	21	register	original	10101
GPR22	GPR	GPR	32	22	register	original	10110
GPR23	GPR	GPR	32	23	register	original	10111

register
zero-register
return register
stack pointer
frame pointer
link register
program counter
instruction registe
instruction memory
data memory
carry-flag
overflow-flag
zero-flag
negative-flag
loop counter

Figure 14: Expanded storage list

4.2. Register specification setting

When you select [Register] tab in [Storage Spec] window, you can define the specification of the registers (see Figure 15).



Storage Name	Resource	Bit Width	Usage	Location
PC	PC	32	program counter	original
IR	IR	32		original

register
zero-register
return register
stack pointer
frame pointer
link register
program counter
instruction register
instruction memory
data memory
carry-flag
overflow-flag
zero-flag
negative-flag
loop counter

Figure 15: Register specification

Table 7 shows the register specification.

Table 7: Specification of register

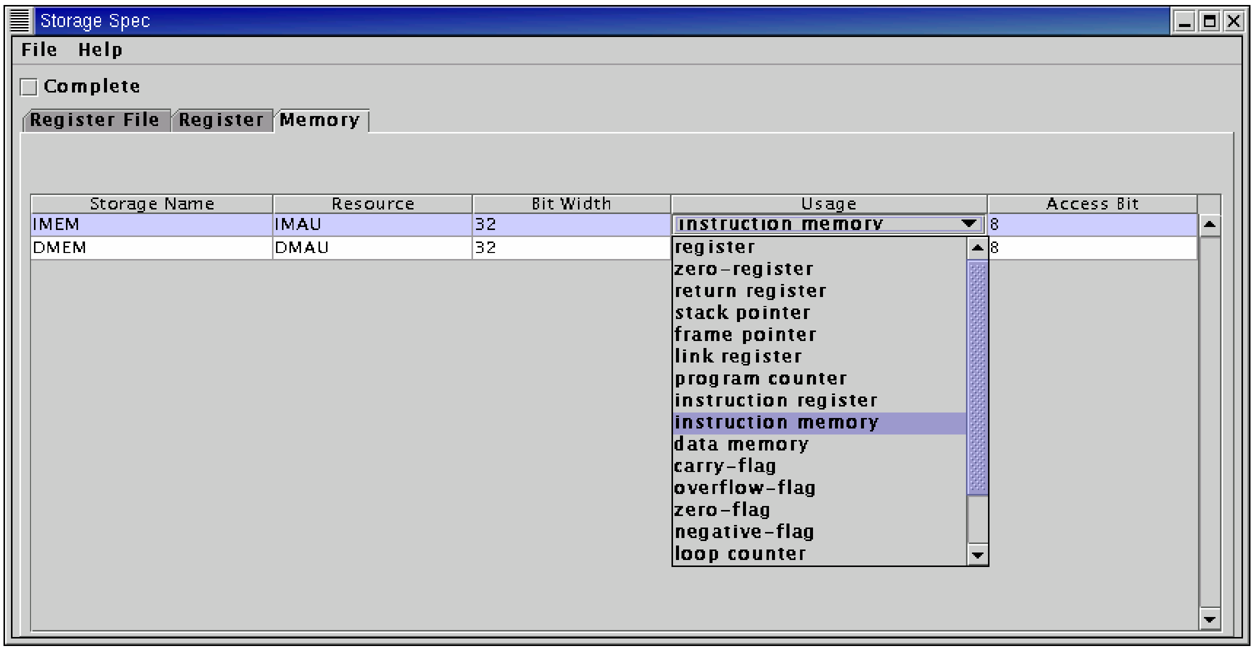
Specification	Description
Storage Name	Storage name used in assembly code
Resource	Resource name declared in [Resource Declaration] step
Bit Width	Storage data bit width
Usage	Storage usage that specify how to handle the storage in the compiler
Location	Storage location

The policy of [Usage] is same as the register file specification. Please refer Table 6.

You can specify overlapped register in location field. When designers would like to specify overlapped register, they describe storage name list to the location field. For example, if you have two registers HI and LO, you can specify overlapped register HL. Clicking [ADD] button, new line appears in the window. Set the name for the overlapped register (like HL) in [Storage Name], resource name (like HI & LO) in [Resource], total bit-width in [bit Width], usage in [Usage] and location (like HI, LO) in [Location]. If not necessary, just write "original".

4.3. Memory specification setting

When you select [Memory] tab in [Storage Spec] window, you can define the specification of the memories (see Figure 16). Table 8 shows the register file specification.



Storage Name	Resource	Bit Width	Usage	Access Bit
IMEM	IMAU	32	instruction memory	8
DMEM	DMAU	32		8

Figure 16: Memory specification

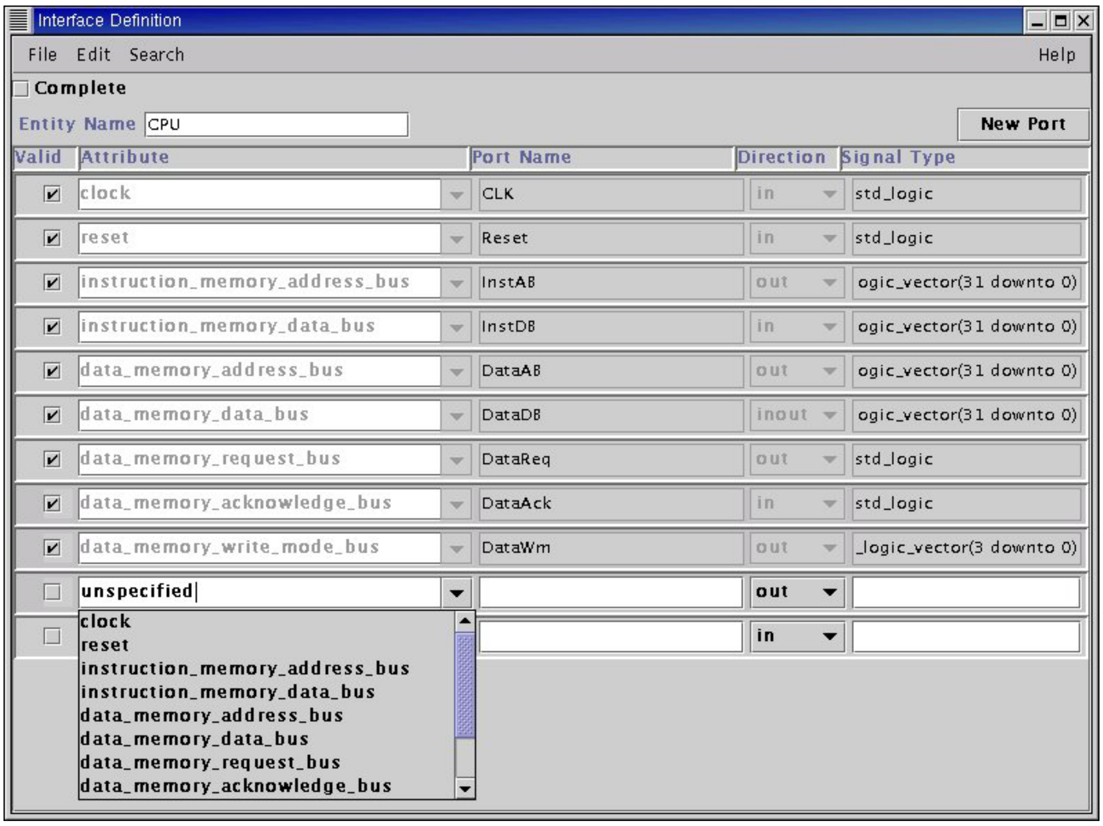
Table 8: Specification of memory

Specification	Description
Storage Name	Storage name used in assembly code
Resource	Resource name declared in [Resource Declaration] step
Bit Width	Storage data bit width
Usage	Storage usage that specify how to handle the storage in the compiler
Access Bit	Minimum access bit width when the processor accesses the memory

The policy of [Usage] and [Location] is same as the register file specification.

5. Interface Definition

[Interface Definition window (Figure 17) opens by clicking [Interface Definition] item in [ASIP Meister Main Menu] window. With this window, you can enter the instance name of the processor and I/O information to determine the external interface of the processor.



☐ Complete

Entity Name: CPU

Valid	Attribute	Port Name	Direction	Signal Type
✓	clock	CLK	in	std_logic
✓	reset	Reset	in	std_logic
✓	instruction_memory_address_bus	InstAB	out	ogic_vector(31 downto 0)
✓	instruction_memory_data_bus	InstDB	in	ogic_vector(31 downto 0)
✓	data_memory_address_bus	DataAB	out	ogic_vector(31 downto 0)
✓	data_memory_data_bus	DataDB	inout	ogic_vector(31 downto 0)
✓	data_memory_request_bus	DataReq	out	std_logic
✓	data_memory_acknowledge_bus	DataAck	in	std_logic
✓	data_memory_write_mode_bus	DataWm	out	_logic_vector(3 downto 0)
☐	unspecified		out	
☐			in	

clock
reset
instruction_memory_address_bus
instruction_memory_data_bus
data_memory_address_bus
data_memory_data_bus
data_memory_request_bus
data_memory_acknowledge_bus

Figure 17: [Interface Definition] window

With ASIP Meister, only the instruction memory access unit (IMAU) and the data memory access unit (DMAU) can connect to the external ports. According to the port attributes, HDL generation engine connects I/O ports and IMAU and DMAU ports.

- **[Entity Name]: Circuit name**

This field shows the instance name of the target processor. The HDL generation engine uses the name.
The name applies to the instance name of the processor description generated in [HDL Generation] step. You must not use reserved words described in appendix section 4.1 and 4.2 as the entity name of the target processor.
The name with ".vhd" extension becomes the file name of main VHDL description.

- **[Valid]: Port Valid/Invalid setting**

This option specifies to implement the port or not to implement. When the checkbox is checked, the port is implemented in the processor. When the checkbox is unchecked, the port is not implemented as an external port of the processor.

- **[Attribute]: Attribute of the port**

This field specifies the port attribute. According to the attributes specified here, HDL generation engine connects the port and the internal signal line. The Table 9 lists the attributes and the descriptions.

Table 9: External port attribute list

Attribute	Description
clock	Connected with clock port of each resource
reset	Connected with reset port of each resource
instruction_memory_address_bus	IMAU address bus
instruction_memory_data_bus	IMAU data bus
data_memory_address_bus	DMAU address bus
data_memody_data_bus	DMAU data bus
data_memory_request_bus	DMAU request bus
data_memory_acknowledge_bus	DMAU acknowledge bus
data_memory_write_mode_bus	DMAU writing mode specifying bus
interrupt	External interrupt input port
unspecified	User defined port

The user-defined port is connected with the “wire_in”, “wire_out” or “wire_inout” resource that its instance name is same as the port name.

- **[Port Name]: Port name**

You cannot change the port name when the checkbox in [Valid] field is checked. To change the external port name, first uncheck the checkbox and change the name. You must not use reserved words described in appendix section 4.1 and 4.2 as the port name.

- **[Direction]: Direction of the signal**

Pull down the menu to select the direction of the external signal. You can select [in] for input signal, [out] for output signal and [inout] for input and output signal.

- **[Signal Type]: Type of the port**

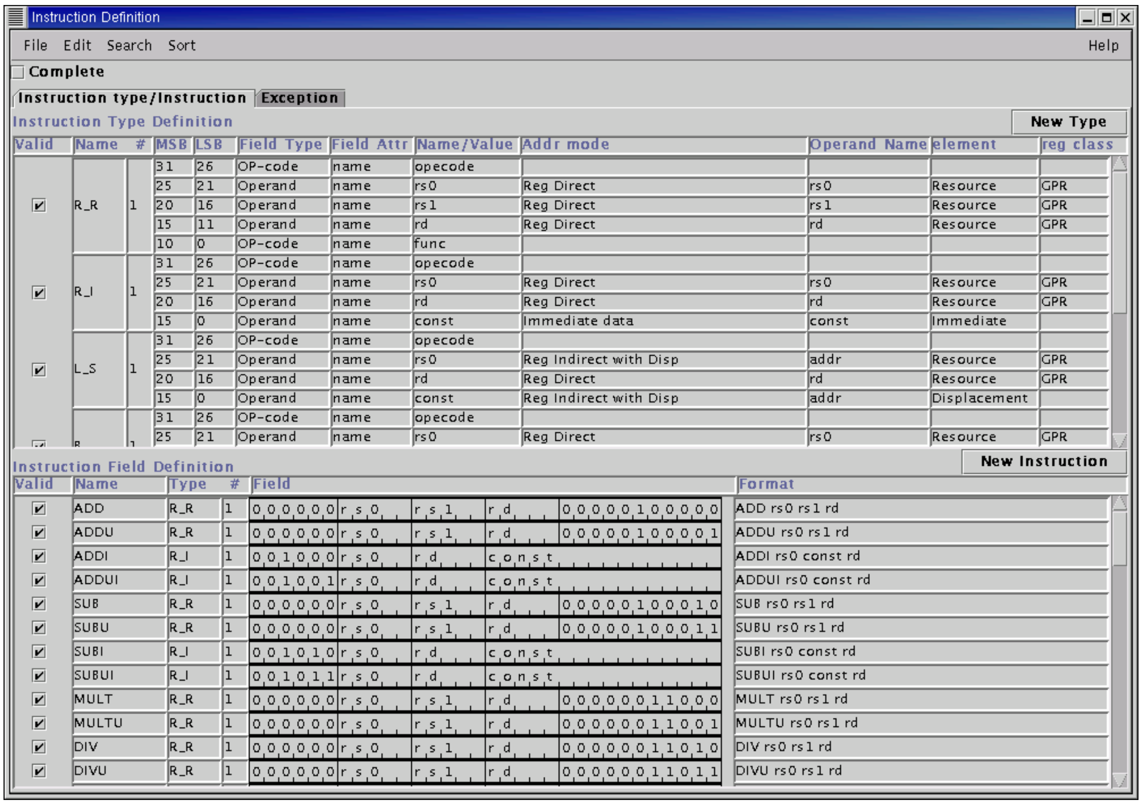
This field specifies the type of the port. You can specify the type declared in the IEEE std_logic_1164 package.

- **[New Port]: Adding port**

Clicking [New Port] button, a new blank row appears at the end of the list to declare a new port.

6. Instruction Definition

[Instruction Definition] window (Figure 18) opens by clicking [Instruction Definition] item in [ASIP Meister Main Menu] window.



Instruction Type Definition

Valid	Name	#	MSB	LSB	Field Type	Field Attr	Name/Value	Addr mode	Operand Name	element	reg class
✓	R_R	1	31	26	OP-code	name	opecode				
			25	21	Operand	name	rs0	Reg Direct	rs0	Resource	GPR
			20	16	Operand	name	rs1	Reg Direct	rs1	Resource	GPR
			15	11	Operand	name	rd	Reg Direct	rd	Resource	GPR
			10	0	OP-code	name	func				
✓	R_I	1	31	26	OP-code	name	opecode				
			25	21	Operand	name	rs0	Reg Direct	rs0	Resource	GPR
			20	16	Operand	name	rd	Reg Direct	rd	Resource	GPR
			15	0	Operand	name	const	Immediate data	const	Immediate	
✓	L_S	1	31	26	OP-code	name	opecode				
			25	21	Operand	name	rs0	Reg Indirect with Disp	addr	Resource	GPR
			20	16	Operand	name	rd	Reg Direct	rd	Resource	GPR
			15	0	Operand	name	const	Reg Indirect with Disp	addr	Displacement	
✓	R	1	31	26	OP-code	name	opecode				
			25	21	Operand	name	rs0	Reg Direct	rs0	Resource	GPR

Instruction Field Definition

Valid	Name	Type	#	Field	Format
✓	ADD	R_R	1	000000 rs0 rs1 rd 00000100000	ADD rs0 rs1 rd
✓	ADDU	R_R	1	000000 rs0 rs1 rd 00000100001	ADDU rs0 rs1 rd
✓	ADDI	R_I	1	001000 rs0 rd const	ADDI rs0 const rd
✓	ADDUI	R_I	1	001001 rs0 rd const	ADDUI rs0 const rd
✓	SUB	R_R	1	000000 rs0 rs1 rd 00000100010	SUB rs0 rs1 rd
✓	SUBU	R_R	1	000000 rs0 rs1 rd 00000100011	SUBU rs0 rs1 rd
✓	SUBI	R_I	1	001010 rs0 rd const	SUBI rs0 const rd
✓	SUBUI	R_I	1	001011 rs0 rd const	SUBUI rs0 const rd
✓	MULT	R_R	1	000000 rs0 rs1 rd 00000011000	MULT rs0 rs1 rd
✓	MULTU	R_R	1	000000 rs0 rs1 rd 00000011001	MULTU rs0 rs1 rd
✓	DIV	R_R	1	000000 rs0 rs1 rd 00000011010	DIV rs0 rs1 rd
✓	DIVU	R_R	1	000000 rs0 rs1 rd 00000011011	DIVU rs0 rs1 rd

Figure 18: [Instruction Definition] window

With [Instruction Definition] window, you can define the instruction types, instruction formats and interrupt conditions. To define the instruction type and instruction format, click [Instruction type/instruction] tab. You cannot define the instruction format without the instruction type, so, you must define the instruction type first. To define the interrupt, click [Exception] tab.

6.1. Instruction type definition

This section explains how to define instruction types.

6.1.1. [Instruction Type Definition] field

[Instruction Type Definition] field describes the instruction type information with the following values;

- **[Valid]: valid/invalid the instruction type**

This checkbox specifies to allow using the instruction type or not. Check the checkbox to allow using the instruction type and uncheck it not to allow using the instruction type.

The instruction type with the same name but a different identification number ([#]) cannot be used simultaneously. If you try to check [Valid] checkbox of instruction type “IT (2)” with some instructions with the instruction type “IT (1)” used, a confirmation dialog appears. When you select [OK] in the dialog, the [Valid] checkbox of all instructions with the instruction type “IT (1)” becomes unchecked (not to implement).

- **[name]: name of the instruction type, [#]: identification number of the instruction type**

This field shows the name and the identification number of the instruction type. The instruction types with same name and different identification number are regarded as different instruction types.

- **[MSB]: most significant bit and [LSB]: least significant bit of field of the instruction type filed**

These fields display the most significant bit (MSB) and the least significant bit of each instruction type field.

- **[Field Type]: type and [Field Attr]: share/un-sharing of the instruction type field**

[Field Type] box displays the type of the each field and [Field Attr] box shows the name or the binary value of the field. Table 10 describes [Field Type] and [Field Attr] pair.

Table 10: [Field Type] and [Field Attr]

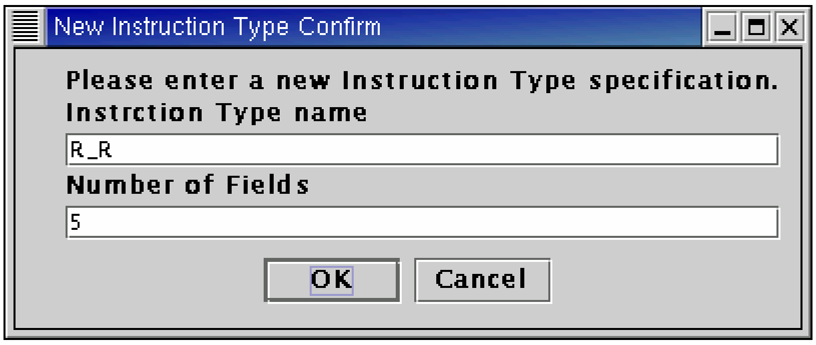
Field Type	Available [Field Attr]	Description
OP-code	binary	If you would like to use the same value for all instructions of defining instruction type, select this pair.
	name	If you would like to set different value in the field for each instruction of defining instruction type, select this pair.
Operand	name	If you would like to specify the field as operand, select this pair.
Reserved	binary	If you would like to reserve the field for future extension and this field must be the specific binary, select this pair.
	don't_care	If you do not mind what value appears in this field, select this pair.

- **[Name/Value]: Name/value of field**

This field displays the name or value of the field. When [binary] is displayed in [Field Attr], this field should be filled with the binary value, and when [name] is displayed, this field displays the name.

6.1.2. Instruction type definition

[Instruction Type Definition] field has [New Type] button. Clicking [New Type] button, [New Instruction Type confirm] window (Figure 19) will open. In [New Instruction Type confirm] window, enter the instruction type name and the number of fields. You must not use reserved words described in appendix section 4.1 as the instruction type name.


New Instruction Type Confirm

Please enter a new Instruction Type specification.

Instrction Type name

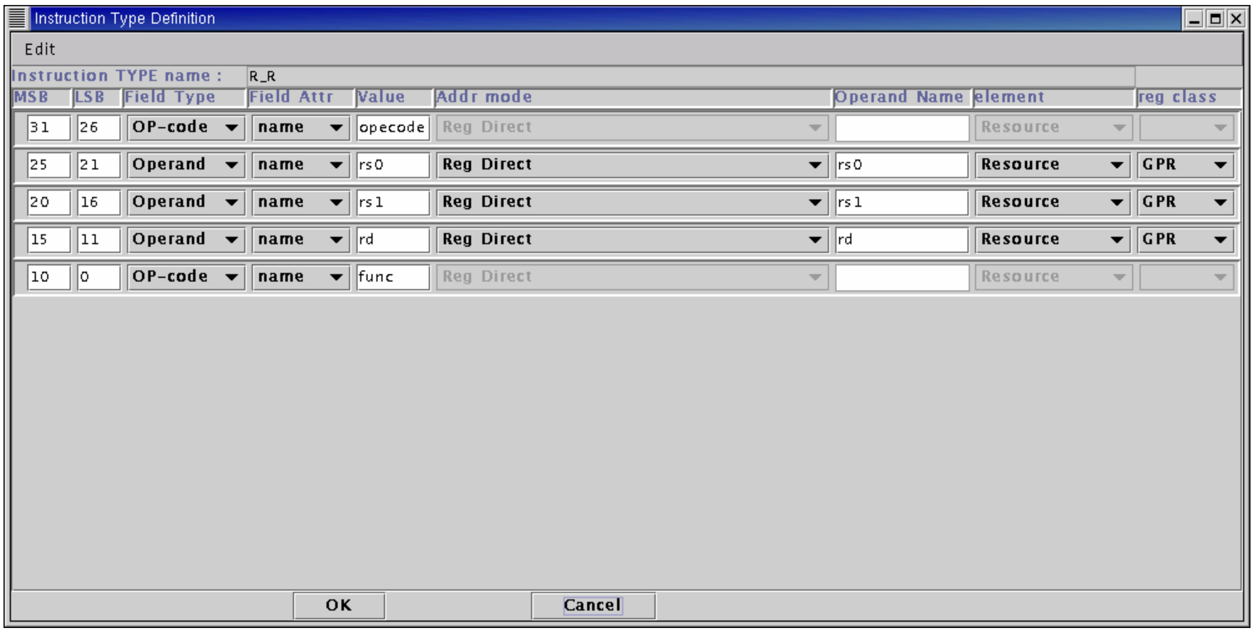
R_R

Number of Fields

5

Figure 19: [New Instruction Type Confirm] window

If you click [OK] button in [New Instruction Type Confirm] window, [Instruction Type Declaration Window] window (Figure 20) opens.



Instruction TYPE name : R_R

MSB	LSB	Field Type	Field Attr	Value	Addr mode	Operand Name	element	reg class
31	26	OP-code	name	opecode	Reg Direct		Resource	
25	21	Operand	name	rs0	Reg Direct	rs0	Resource	GPR
20	16	Operand	name	rs1	Reg Direct	rs1	Resource	GPR
15	11	Operand	name	rd	Reg Direct	rd	Resource	GPR
10	0	OP-code	name	func	Reg Direct		Resource	

Figure 20: [Instruction Type Declaration Window] window

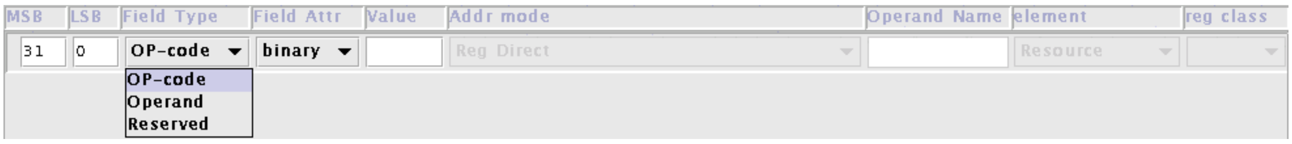
Use this window to define the fields. The fields in this window show the following information. After you have set all settings of the defining instruction type, please click [OK] button to return to [Instruction Definition] window (Figure 18).

- **Field bound: [MSB],[LSB]**

Enter the MSB and LSB of each field.

- **Field type: [Field Type]**

Select the field type from the pull down menu (Figure 21). That menu consists of [Op-code] (operation code), [Operand] (operand) and [Reserved] (reserved field).

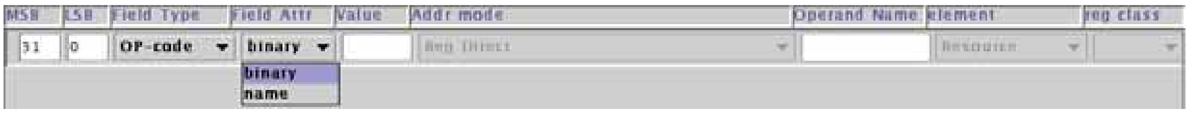


MSB	LSB	Field Type	Field Attr	Value	Addr mode	Operand Name	element	reg class
31	0	OP-code	binary		Reg Direct		Resource	

Figure 21: [Field Type] pull down menu

- **Field attribute: [Field Attr]**

Select the field attribute from the pull down menu (Figure 22). Table 10 shows the [Field Type] and [Field Attr] relations.



MSB	LSB	Field Type	Field Attr	Value	Addr mode	Operand Name	element	reg class
31	0	OP-code	binary		Reg Direct		Resource	

Figure 22: [Field Attr] pull down menu

- **Field value: [Value]**

Set the field value. If [Field Attr] is [binary], set the binary value, and if [Field Attr] is [name], set the field name. You must not use reserved words described in appendix section 4.1 as the instruction field name.

- **[Addr Mode] and [element]: Addressing mode**

Operand usage is specified with the addressing mode. Table 11 describes the addressing modes.

Table 11: Addressing mode

Addressing Mode	Type of elements	Description
Reg Direct	Resource	The data is in the register specified by the value.
Indirect	Resource	The data is in the register specified by the data of the register specified by the value.
Reg Indirect	Resource	The data is in the memory specified by the data of the register specified by the value.
Reg Indirect with Pre-decrement	Resource	The data is in the memory specified by the data of the register specified by the value. The data of the register is decremented before memory access.
Reg Indirect with Pre-increment	Resource	The data is in the memory specified by the data of the register specified by the value. The data of the register is incremented before memory access.
Reg Indirect with Post-decrement	Resource	The data is in the memory specified by the data of the register specified by the value. The data of the register is decremented after memory access.
Reg Indirect with Post-increment	Resource	The data is in the memory specified by the data of the register specified by the value. The data of the register is incremented after memory access.
Reg Indirect with Disp	Resource and Displacement	The data is in the memory specified by the data of the register specified by the value. The data of the register is displaced by the value after memory access.
Reg Indirect with Disp and increment	Resource and Displacement	The data is in the memory specified by the data of the register specified by the value. The data of the register is displaced by the value and incremented after memory access.
Reg Indirect with Index	Resource and Displacement	The data is in the memory that address equals to the sum of the data of the register specified by the value and the index value.
Reg Indirect with Index and increment	Resource and Displacement	The data is in the memory that address equals to the sum of the data of the register specified by the value and the index value. The data of the register is incremented after memory access.
Reg Indirect with Scaled Index	Resource and Displacement	The data is in the memory that address equals to the sum of the data of the register specified by the value and the scaled index value.
Reg Indirect with Disp and Scaled Index	Resource and Displacement	The data is in the memory that address equals to the sum of the data of the register specified by the value and the scaled index value.
PC relative address	Symbol	PC Relative addressing
Absolute address	Symbol	Absolute addressing
Immediate data	Immediate	Immediate data

- **[Operand Name]: Name of the operand**

In the assembler language, operands are specified with mnemonic format. In ASIP Meister, mnemonic format is described using string and operand name. Designers can declare the name of the operand in this field. Please note that when you select the addressing mode that consists of two elements such as [Register Indirect with Displacement] addressing and you would like to make an operand with two fields, please describe the same operand name at "Operand Name" section.

- **[reg class]: Register class**

When you select the type of element as [Resource], you have to specify register class. The register class specifies the accessible registers of the operand. For example, if you select [GPR] from the list, the operand can access [GPR] register file. If you select [GPR0], that is one of the register file elements, the operand can only access [GPR0] register. The register class is declared in the storage declaration step. Note that this register class has been declared at the previous design phase [Storage Specification].

6.2. Instruction definition

This section explains how to define instructions.

6.2.1. [Instruction Field Definition] field

[Instruction Field Definition] field describes the instruction information with the following values;

- **[Valid]: valid/invalid the instruction**

This checkbox specifies to allow using the instruction or not. Check the checkbox to allow using the instruction and uncheck it not to allow using the instruction.

- **[name]: name of the instruction**

This field shows the name of the instruction.

- **[Type], [#]: instruction type**

This field shows the instruction type and the type identification number.

- **[Field]: instruction mnemonic**

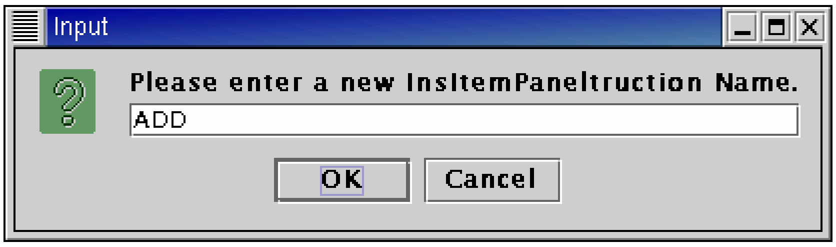
This field shows the mnemonic of the instruction. Each field shows the binary value or the name.

- **[Format]: instruction format**

This field shows the instruction format.

6.2.2. Instruction definition

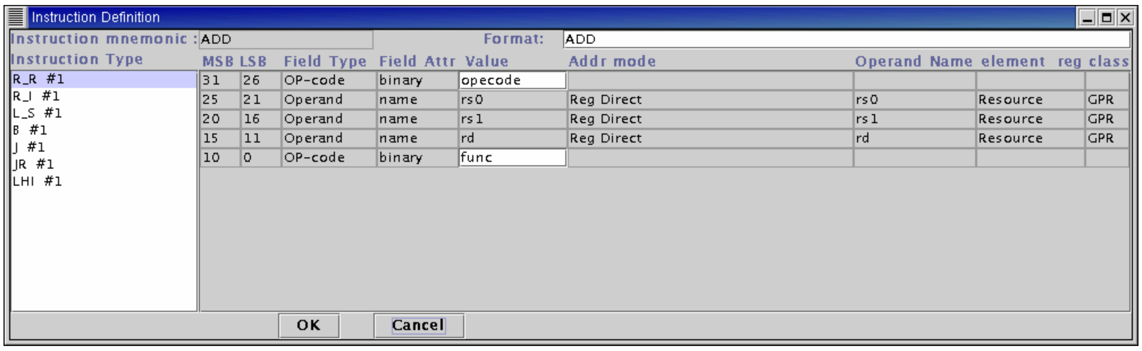
If you click [New Instruction] button on [Instruction Field Definition] field, [Input] window (Figure 23) opens. Enter the instruction name and click [OK] button, and then, [Instruction Definition] window (Figure 24) will open. You must not use reserved words described in appendix section 4.1 as the instruction name.


Input

Please enter a new InsItemPanelItruction Name.

ADD

Figure 23: [Input] window



Instruction mnemonic : ADD   Format: ADD

Instruction Type

R_R #1
R_I #1
L_S #1
B #1
J #1
JR #1
LHI #1

MSB	LSB	Field Type	Field Attr	Value	Addr mode	Operand Name	element	reg class
31	26	OP-code	binary	opecode				
25	21	Operand	name	rs0	Reg Direct	rs0	Resource	GPR
20	16	Operand	name	rs1	Reg Direct	rs1	Resource	GPR
15	11	Operand	name	rd	Reg Direct	rd	Resource	GPR
10	0	OP-code	binary	func				

Figure 24: [Instruction Declaration] window

In [Instruction Definition] window, set the following information;

- **[Instruction mnemonic]: Instruction name**

This field displays the instruction name that has been entered in [input] window.

- **[Instruction Type]: Select the instruction type**

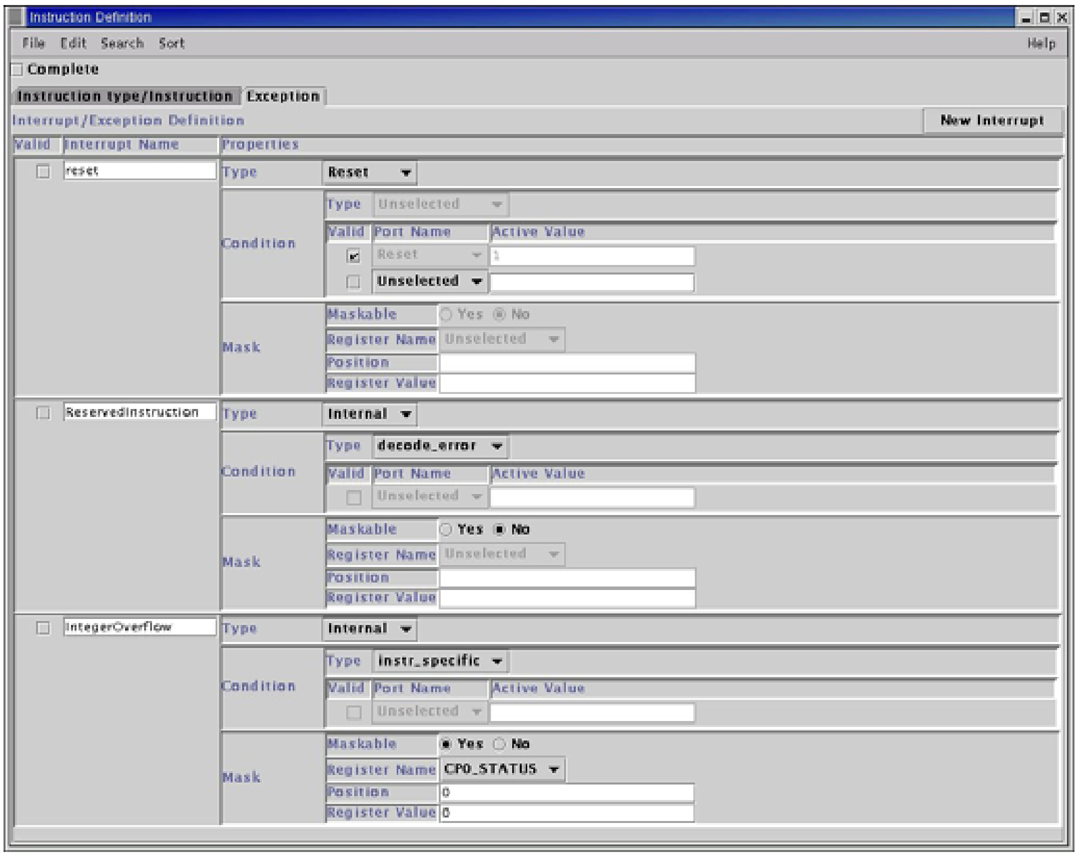
This field lists the defined instruction type and its identification number. Select the instruction type to use from this list. After you select the instruction type, the right pane of the window displays number of rows corresponding to the number of the fields. Note that each row corresponds to each field and each row includes the value of [MSB], [LSB], [Field Type], [Field Attr] and [Value].

- **[Value]: Value of field**

This field displays the value of each field. [Value] field of the instruction is only enabled when [Field Type] and [Field Attr] are set to [OP-code] and [name] respectively in the instruction type definition. Enter the value of the field to [value] option. Otherwise, the field value of the instruction type is inherited.

6.3. Exception definition

This section explains how to define condition of exceptions.
By clicking [Exception] tab in [Instruction Definition] window, you switch to [Interrupt/Exception Declaration] pane.



Valid	Interrupt Name	Properties		
☐	reset	Type	Reset	
		Condition	Type	Unselected
			Valid / Port Name / Active Value	✓ Reset 1; ☐ Unselected
		Mask	Maskable	Yes [✓ No]
			Register Name	Unselected
			Position	
			Register Value	
☐	ReservedInstruction	Type	Internal	
		Condition	Type	decode_error
			Valid / Port Name / Active Value	☐ Unselected
		Mask	Maskable	Yes [✓ No]
			Register Name	Unselected
			Position	
			Register Value	
☐	IntegerOverflow	Type	Internal	
		Condition	Type	instr_specific
			Valid / Port Name / Active Value	☐ Unselected
		Mask	Maskable	[✓ Yes] No
			Register Name	CP0_STATUS
			Position	0
			Register Value	0

Figure 25: [Exception] window

The window consists of the following information;

- **[Valid]: Enable/Disable interrupt**

  Use this checkbox to enable/disable the defined interrupt.

- **[Interrupt Name]: Interrupt (Exception) name**

Set the interrupt or exception name. You must not use reserved words described in appendix section 4.1 as the interrupt or exception name.

- **[Type]: Interrupt type**

Select the interrupt type from [External] for the external interrupt, [Internal] for the internal interrupt and [Reset] for the reset interrupt.

- **[Condition]: Describe the condition of interrupt generation**

Set the conditions of the interrupt.
If it is the internal interrupt, also specify the sub type of the interrupt. Select [decode_error] for an interrupt that catches the instruction decode error, [instr_specific] for an interrupt generated from specified instructions.
For all interrupts, select the port of interrupt signal. [Valid] checkbox enables you to select to implement the interrupt or not. [Port Name] shows the port name that generates the interrupt, and when the port shows the value set in [Active Value], the processor detects the interrupt occurrence.

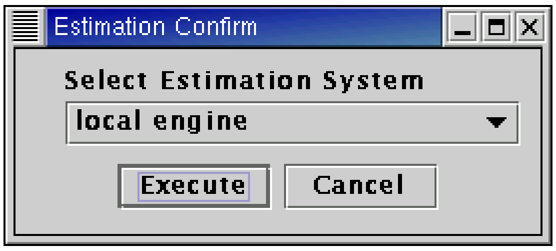
- **[Mask]: Describe a condition of an interrupt mask**

In this field, enter a condition of masking an interrupt.
If you would like to enable the interrupt mask, select [Yes] in [Maskable] radio button.
[Register Name] shows the mask register name and the value of [Position] is the masking bit in the mask register. The interrupt is masked when the masking bit of the mask register equals to the value of [Register Value].
[Register Name] is selected from the pull down menu that shows registers specified as [Mask Register] in [Resource Declaration] window. [Position] is the number from zero to the number that is the bit width of the register minus one. Enter zero if the selected register is a single bit register. [Register Value] is “0” or “1”.
Note that this version of ASIP Meister only supports one external interrupt. Use an interrupt controller outside the processor to use more than one external interrupt.

7. Arch. Level Estimation

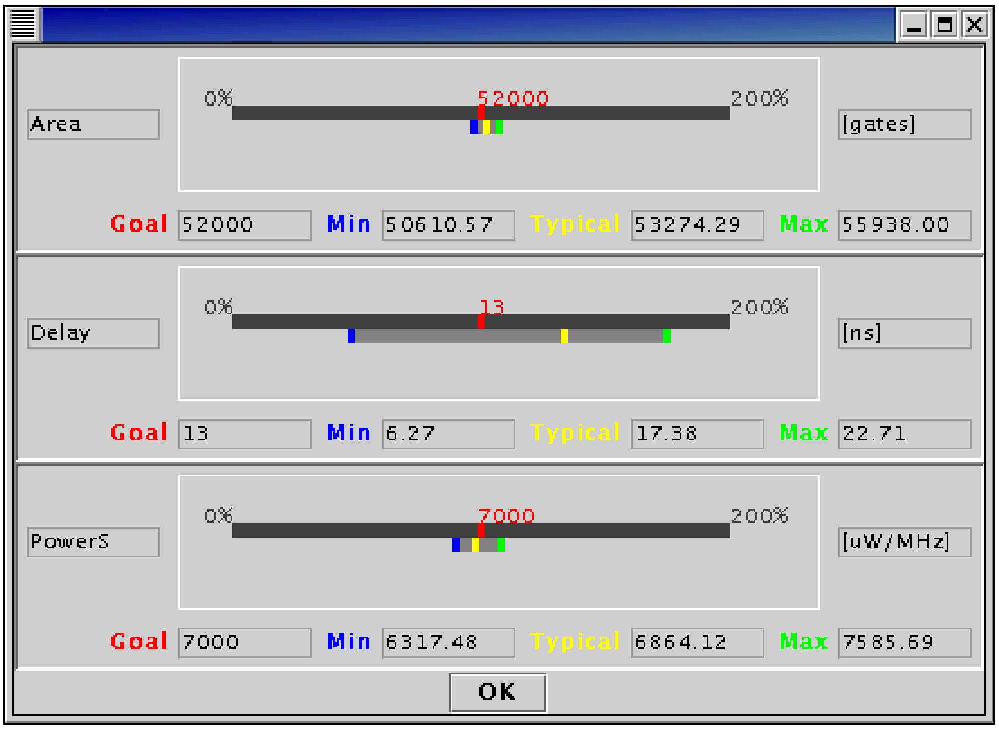
[Estimation Confirm] window (Figure 26) opens by clicking [Arch. Level Estimation] item in [ASIP Meister Main Menu] window. You can check the estimated performance of the processor (area, delay and static power consumption) with this step. Since this is the estimation before finalizing the details of the design, the estimated values are not exact values, but these values can help your following design steps. If the estimation result shows lower value than your expectation, you can change the settings and re-estimate the qualities.

This version only supports estimation using local engine. So, just click [Execute] button and the new window (Figure 27) with the estimated values opens.



Select Estimation System: local engine

Figure 26: [Estimate Confirm] window



		Goal	Min	Typical	Max
Area	[gates]	52000	50610.57	53274.29	55938.00
Delay	[ns]	13	6.27	17.38	22.71
PowerS	[uW/MHz]	7000	6317.48	6864.12	7585.69

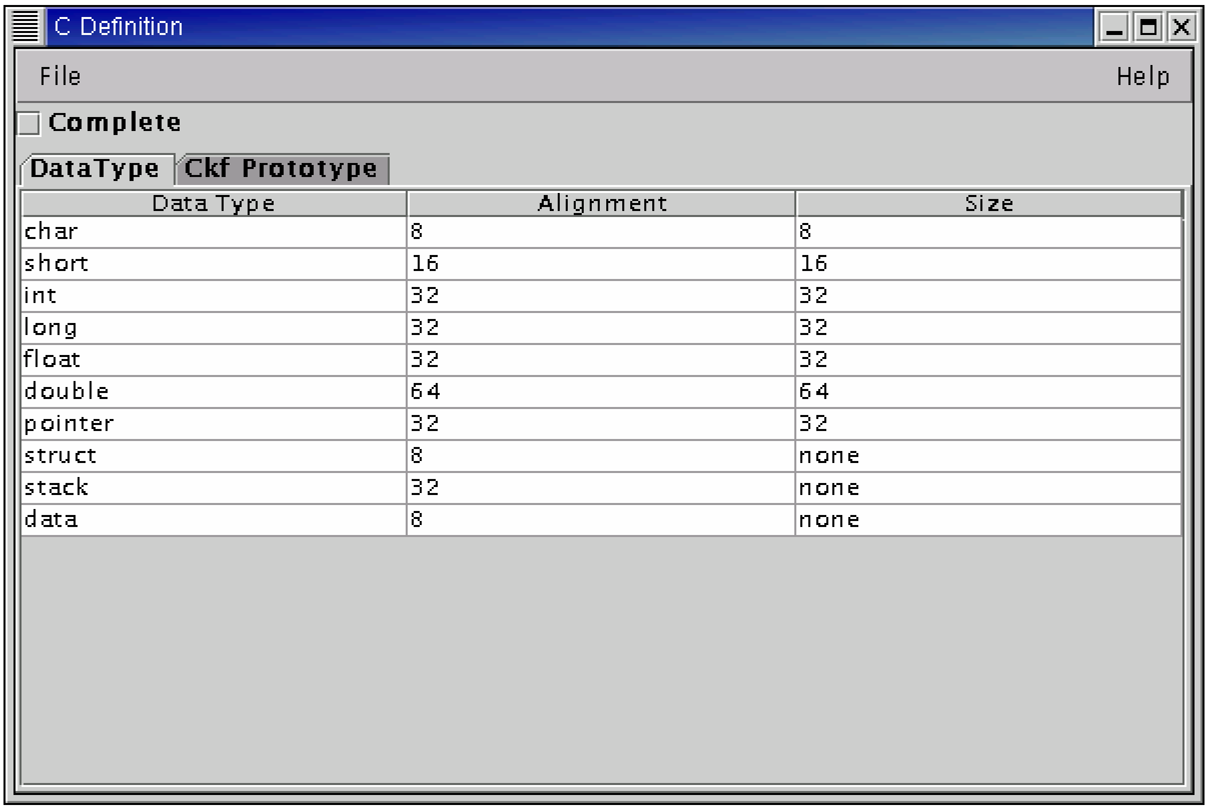
Figure 27: Estimate result window

The window displays the estimation results of area, maximum path delay and static power consumption. [Min] and [Max] show minimum and maximum value respectively. [Typ] shows the value that is estimated with the design priority set in [Design Goal & Arch. Design] step. Middle value with red line shows the design goal set in [Design Goal & Arch. Design] step.

8. C Definition

C definition consists of data type definition, structure declaration and CKF (Compiler-Known-Function) declaration. In data type definition, data alignment and data size can be specified. Compiler generator specifies data alignment using this information, and assembler allocates data using this information.
Compiler-Known-Function is the method to specify special instruction such as DCT instruction, FIR filter instruction and so on.

When you click [C Definition] item in [ASIP Meister Main Menu] window, [C Definition] window opens (Figure 28).



Data Type	Alignment	Size
char	8	8
short	16	16
int	32	32
long	32	32
float	32	32
double	64	64
pointer	32	32
struct	8	none
stack	32	none
data	8	none

Figure 28: [C Definition] window

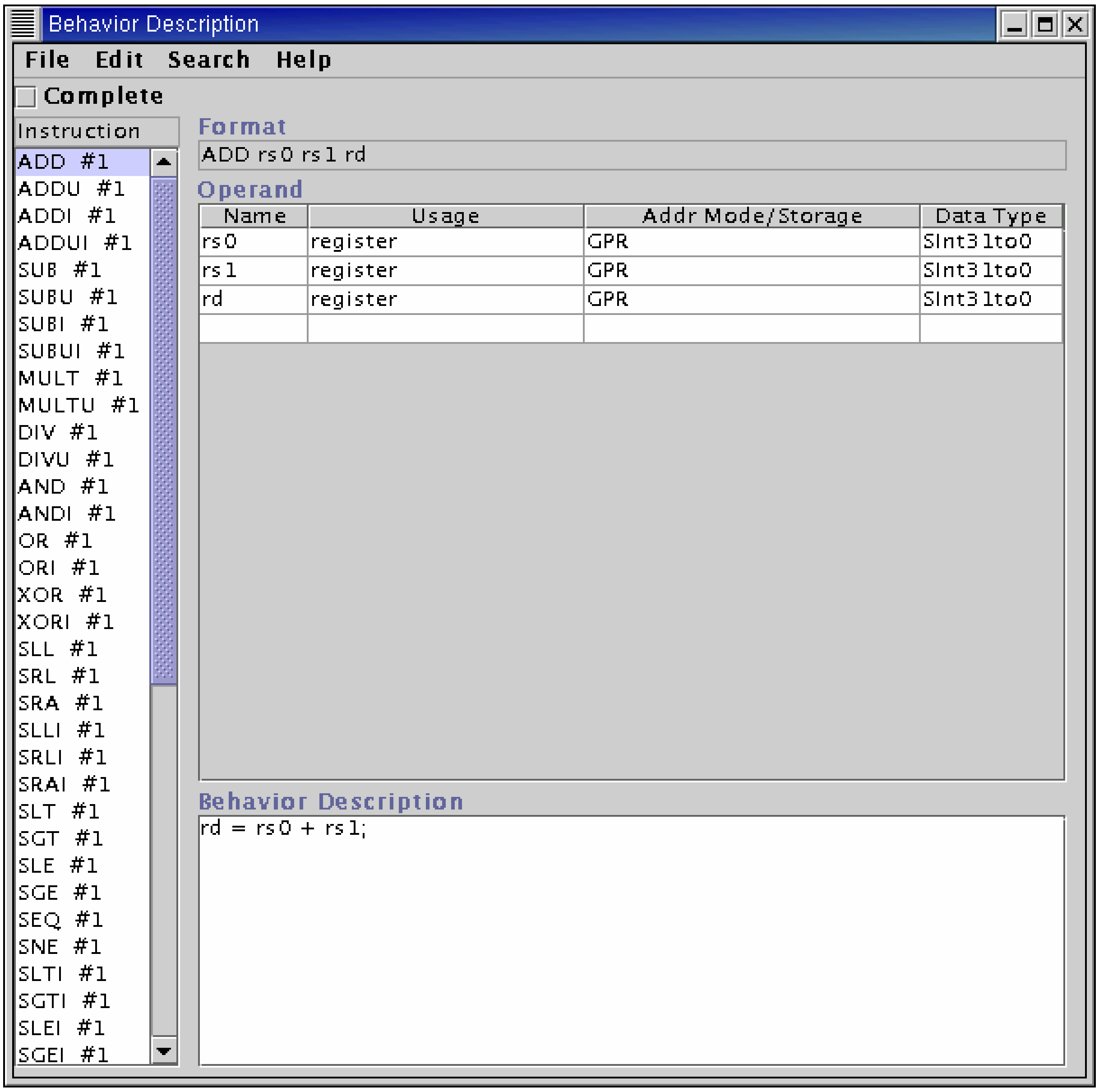
In this part, data alignment and data size are specified. When the window opens, the window shows default value. If necessary, this information will be changed.

When CKF prototype tub is clicked, Compiler-Known-Functions declaration window opens. If the function in C language is described in this window, the compiler can translate the function to the instruction.

9. Behavior Description

The behavior descriptions represent the instruction behavior and we use C like semantics. For each instruction, we declare operands used as variables of behavior descriptions. Operand field includes the operand name, usage, addressing mode, and data type for each operand.

When you click [Behavior Description] item in [ASIP Meister Main Menu] window, [Behavior Description] window opens (Figure 29).



ADD rs0 rs1 rd

Name	Usage	Addr Mode/Storage	Data Type
rs0	register	GPR	SInt31to0
rs1	register	GPR	SInt31to0
rd	register	GPR	SInt31to0

```
rd = rs0 + rs1;
```

Figure 29: [Behavior Description] window

The window shows the operands specified in the instruction declaration step. If necessary, you can add operands as the implicit input/output data. The usages of operands are the same as storage definition part (see Table 6).

[Addr Mode/Storage] shows the addressing mode specified in operand declaration. Addressing mode is listed in Table 12.

Table 12: Addressing mode of operands

Addressing mode	Description
Register_name (eg GPR, GPR0)	Register direct
[register_name, disp]:memory_name (eg [GPR, disp]:DMEM)	Register indirect
@memory_name (eg @DMEM)	Memory direct
@[memory_name] (eg @[DMEM])	Memory indirect
Immediate	Immediate data used for the calculation
Label	Label that indicates jump/branch address (you should define a label for a jump/branch instruction)

[Data Type] shows the data type of the operand. Table 13 shows the data types.

Table 13: Data type of operands

Data type	Description
SInt*X*to0	Signed data that bit width is *X* + 1
UInt*X*to0	Unsigned data that bit width is *X* + 1
Int*X*to0	Signed or unsigned data that bit width is *X* + 1
label	Label operand
any	Do not specify about data type in this operand

9.1. Behavior Semantics

9.1.1. Operations

Table 14 and Table 15shows the operations for behavior description. These operations are based on C language operations.

Table 14: Operations (one operand)

Operation	Example	Description
( )	(*a* + *b*)	Priority specification
~	~ *a*	Not operation
Sign_extended	Sign_extended(*a*)	Sign extension
[bit]	*a*[5]	Bit selection
[a : b]	a[15:0]	Field selection
Next	Next(*a*)	Return address

Table 15: Operations (two operands)

Operation	Example	Description
=	a = b + c	Assignment
+	a + b	Addition
-	a – b	Subtraction
*	a * b	Multiplication
/	a / b	Division
%	a = b % c;	Modulo
&	a = b & c;	And
|	a = b | c;	Or
^	a = b ^ c;	Exclusive or
<<	a = b << c;	Shift left
>>>	a = b >>> c;	Logical shift right
>>	a = b >> c;	Arithmetic shift right
<	a < b	Less than
>	a > b	Greater than
<=	a <= b	Less equal
>=	a >= b	Greater equal
!=	a != b	Not equal
==	a == b	Equal
(a, b)	(a, b) = c * d;	Concatenation
Compare	Compare(a, b)	Comparison

9.1.2. Sentences

In behavior description, the following sentences can be described.

- **Assignment Sentence**

Assignment sentence is basic sentence of the proposed behavior description. It represents data transfer. Data transfer can be categorized as follows; register-to-register transfer, register-to-memory transfer, memory-to-register transfer, and memory-to-memory transfer. Register-to-register transfer instructions include arithmetic instructions such as addition, and multiplication. Register-to-memory transfer instructions include store instructions. Memory-to-register transfer instructions include load instruction.
Moreover, assignment to program counter means jump or branch instructions.

- **Conditional Sentence**

Conditional sentence can represent conditional branch or conditional execution. Conditional sentence includes two parts: condition part and execution part. In condition part, designers can specify condition for instruction execution. For instance, “a == 0” means that if a == 0, then sentence body is executed. Moreover, condition code checking can be specified in the condition part. For example, if you would like to describe zero flag and negative flag checking to represent less than or equal to zero, you can describe as follows;

```
if ( ( Z == “1”) || ( N == “1” ) ) {
   // do something
}
```

Here, Z is zero flag and N is negative flag. ”1” is that the flag is true.
In this version, conditional execution is not supported in compiler generation.

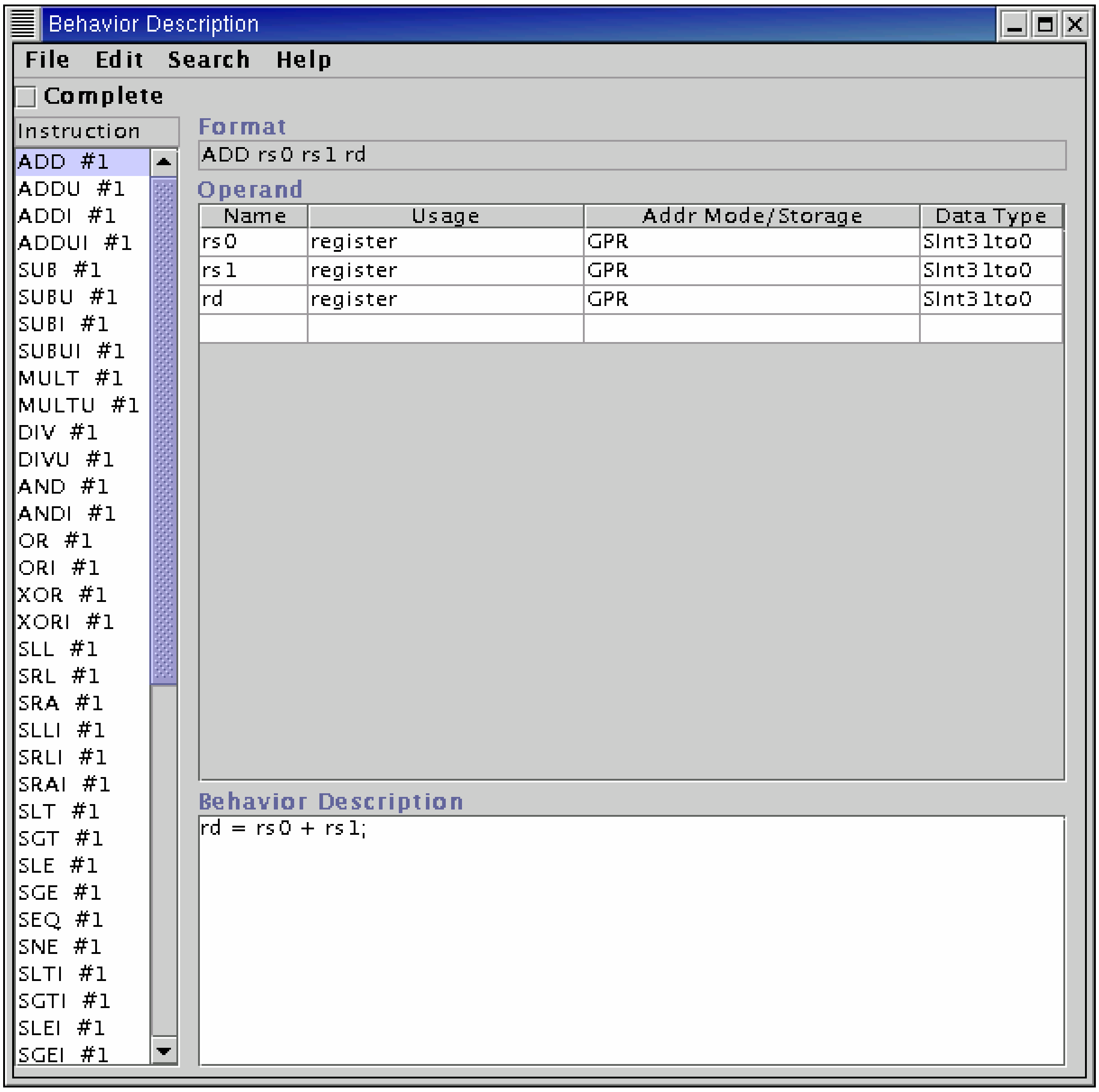
- **Compiler-Known-Functions**

Compiler-Known-Functions can be used for special instructions. Special instructions such as DCT instruction, SIMD instruction and so on, improve target application performance. In C source code, designers specify an instruction using the function. In the generated compiler, the function maps to the instruction in compilation phase. Then, assembly code is emitted.

9.2. Instruction behavior example

9.2.1. Arithmetic Instruction

Arithmetic instructions can be described using C language operations such as “+”, “-”, “*”, “/” and so on. For example, add instruction that reads two operands from registers and write back to one register is described bellow.



ADD rs0 rs1 rd

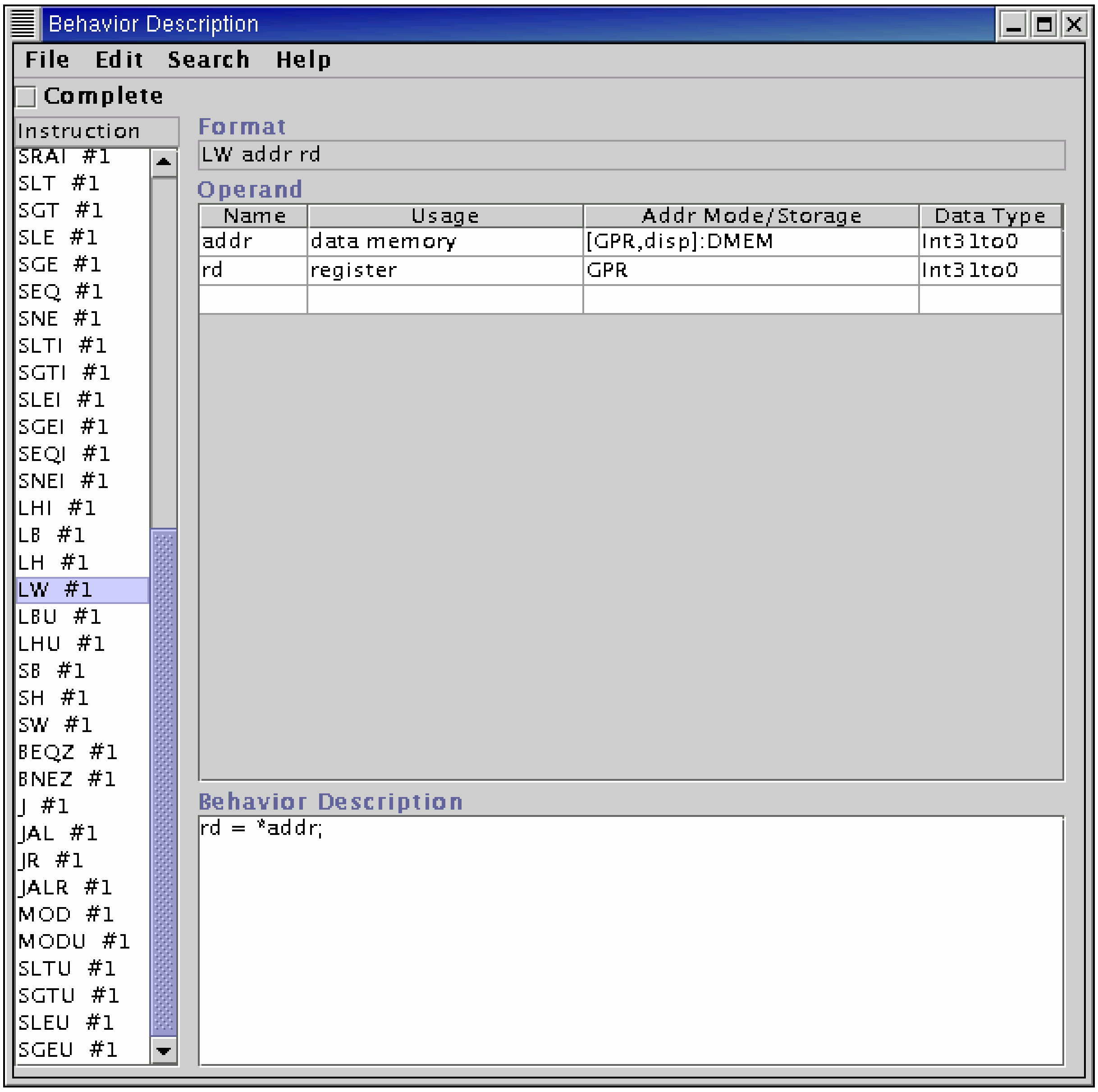
Name	Usage	Addr Mode/Storage	Data Type
rs0	register	GPR	SInt31to0
rs1	register	GPR	SInt31to0
rd	register	GPR	SInt31to0

```
rd = rs0 + rs1;
```

Figure 30: Add instruction.

9.2.2. Load/Store instructions

Load/Store instructions can be described using data assignment operation “=” and memory access addressing mode such as register indirect. If designers describe data transfer from register to memory, this instruction is store instruction. On the contrary, if designers describe data transfer from memory to register, this instruction is load instruction. In Figure 31, register operand and data memory operand are specified. The behavior of "LW" is that data is read from memory "DMEM", and written to register "rd".



LW addr rd

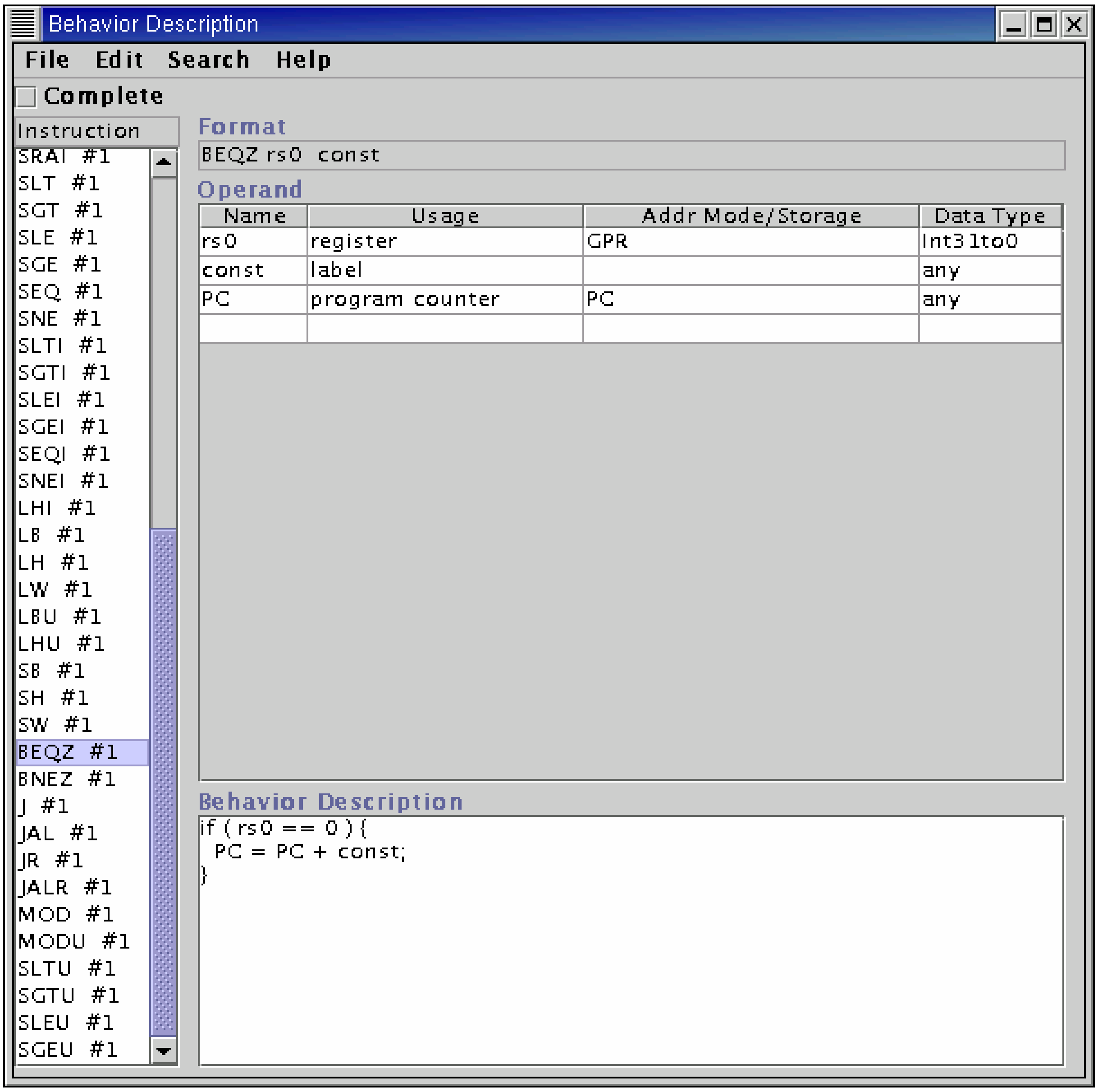
Name	Usage	Addr Mode/Storage	Data Type
addr	data memory	[GPR,disp]:DMEM	Int31to0
rd	register	GPR	Int31to0

```
rd = *addr;
```

Figure 31: Load instruction

9.2.3. Jump and Branch Instructions

Branch instructions can be described using data assignment of program counter. If an instruction is described using conditional operation, the instruction stands for branch instruction. If an instruction is not described by using conditional operation, the instruction stands for jump instruction. In Figure 32, the behavior of "BEQZ" is that comparison between registers "rd" and "rd" is condition of branch.



```
BEQZ rs0 const
```

Name	Usage	Addr Mode/Storage	Data Type
rs0	register	GPR	Int31to0
const	label		any
PC	program counter	PC	any

```
if ( rs0 == 0 ) {
  PC = PC + const;
}
```

Figure 32: Branch instruction

9.3. Compiler-Known-Functions

Compiler-Known-Functions are specified special instructions. CKFs are declared in "C Definition". If you would like to use CKFs, it is required to declare CKFs.

9.4. BNF of Behavior Description

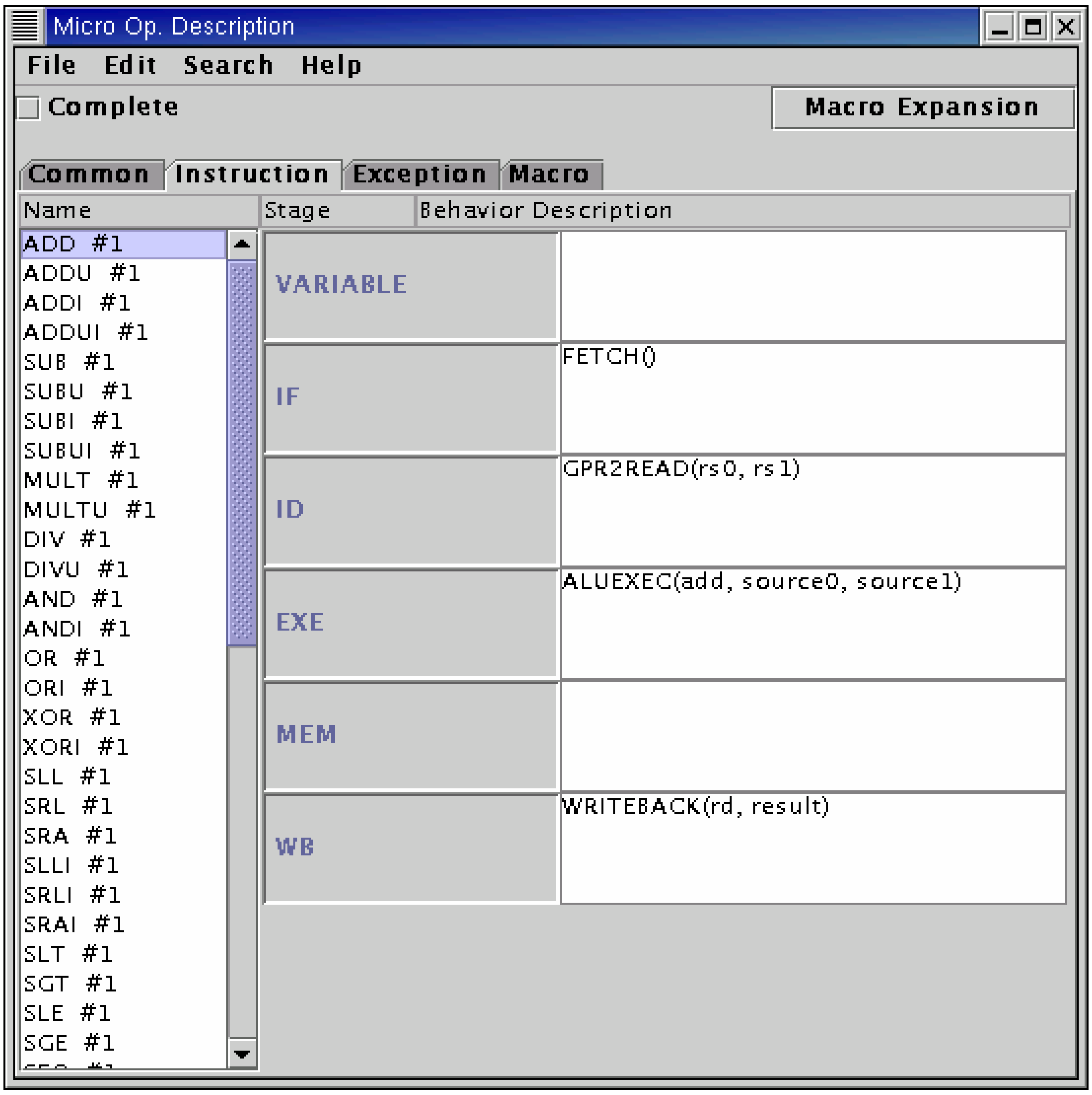
```
<behavior description>    ::=  <normal operation>  |  <control operation>  |
                                 <compare operation>  |  <zol operation  |
                                 <ckf operation>
<normal operation>        ::=  <expression>  {  <expression>  }
<expression>              ::=  [  <left term>  "="  ]  <right term>  [ <set flag> ]  ";"
```

```
<left term>             ::=  [  "*"  ]  <parameter>  |
                               "("  ["*"]  <parameter>  {  ","  ["*"]  <parameter>  }  ")"
<right term>            ::=  <term layer9>  {  "|"  <term layer9>  }
<term layer9>           ::=  <term layer8>  {  "^"  <term layer8>  }
<term layer8>           ::=  <term layer7>  {  "&"  <term layer7>  }
<term layer7>           ::=  <term layer6>  {  "=="  <term layer6>  |  "!="  <term layer6>  }
<term layer6>           ::=  <term layer5>
                               {  "<"  <term layer5>  |  "<="  <term layer5>  |
                                  ">"  <term layer5>  |  ">=" <term layer5>  }
<term layer5>           ::=  <term layer4>
                               {  "<<"  <term layer4>  |
                                  ">>"  <term layer4>  |
                                  ">>>"  <term layer4>  }
<term layer4>           ::=  <term layer3>  {  "+"  <term layer3>  |  "-"  <term layer3>  }
<term layer3>           ::=  <term layer2>
                               {  "*"  <term layer2>  |
                                  "/"  <term layer2>  |
                                  "%"  <term layer2>  }
<term layer2>           ::=  <term layer1>  |
                               "~"  <term layer1>  |
                               "-"  <term layer1>  |
                               "Sign_extended"  "("  <term layer1>  ")"  |
                               "Next"  "("  <term layer1>  ")"
<term layer1>           ::=  "("  <right term>  ")"
                                 [  "["  <positive integer>  ":"  <non_negative integer>  "]"  ]  |
                               "("  <right term>  ")"  [  "["  <non_negative integer>  "]"  ]  |
                               [  "*"  ]  <parameter>
                                 [  "["  <positive integer>  ":"  <non_negative integer>  "]"  ]  |
                               [  "*"  ]  <parameter>
                                 [  "["  <non_negative integer>  "]"  ]  |
                               <constant>
<constant>              ::=  <non_negative integer>
<set flag>              ::=  "{"  <flag instance>  {  ","  <flag instance>  }  "}"
<control operation>     ::=  "if"  <conditions>  "{"  <normal operation>  "}"
                               [  "else"  "{"  <normal operation>  "}"  ]
<conditions>            ::=  "("  <normal conditions>  ")"  |  "("  <flag conditions>  ")"
<normal conditions>     ::=  <parameter>  <comparator sign>  <parameter> |
                               <parameter>  <comparator sign>  <non_negative integer>  |
                               "always"
<comparator sign>       ::=  "=="  |  "!="  |  "<"  |  "<="  |  ">"  |  ">="
<flag conditions>       ::=  <flag condition>
                               {  <logical operator>  <flag condition>  }
<flag condition>        ::=  <flag instance>  "=="  <flag value>
<flag instance>         ::=  <name>
<flag value>            ::=  "0"  |  "1"
<logical operator>      ::=  "&&"  |  "||"
<compare operation>     ::=  "Compare"  "("  <parameter>  ","  <parameter>  ")"  <set flag>  ";"

<zol operation>         ::=  <parameter>  "="  <zol function>  "("  <parameter>  ")"  ";"  |
                               "loop_start"  "("  ")"  ";"
<zol function>          ::=  "start_set"  |  "end_set"  |  "iter_set"  |  "inst_num_set"

<ckf operation>         ::=  [  <left term>  "="  ]  <ckf term>  [  <set flag>  ]  ";"
<ckf term>              ::=  <ckf name>  "("  [  <parameter>  {  ","  <parameter>  }  ]  ")"
```

```
<ckf name>                ::=  <name>
<parameter>               ::=  [  "*"  ]  <alphabets and numbers>

<binary>                  ::=  <bit number>  {  <bit number>  }
<name>                    ::=  <alphabet>  {  <alphabet>  |  <number>  |  "_"  }
<positive integer>        ::=  <top number>  {  <number>  }
<non_negative integer>    ::=  <positive integer>  |  "0"
<alphabet>                ::=  "a"  |  "b"  |  "c"  |  "d"  |  "e"  |  "f"  |  "g"  |
                                "h"  |  "i"  |  "j"  |  "k"  |  "l"  |  "m"  |  "n"  |
                                "o"  |  "p"  |  "q"  |  "r"  |  "s"  |  "t"  |  "u"  |
                                "v"  |  "w"  |  "x"  |  "y"  |  "z"  |
                                "A"  |  "B"  |  "C"  |  "D"  |  "E"  |  "F"  |  "G"  |
                                "H"  |  "I"  |  "J"  |  "K"  |  "L"  |  "M"  |  "N"  |
                                "O"  |  "P"  |  "Q"  |  "R"  |  "S"  |  "T"  |  "U"  |
                                "V"  |  "W"  |  "X"  |  "Y"  |  "Z"
<number>                  ::=  "0"  |  "1"  |  "2"  |  "3"  |  "4"  |
                                "5"  |  "6"  |  "7"  |  "8"  |  "9"
<top number>              ::=  "1"  |  "2"  |  "3"  |  "4"  |  "5"  |  "6"  |  "7"  |  "8"  |  "9"
<binary number>           ::=  "0"  |  "1"
```

10. Micro Op. Description

[Micro Op. Description] window (Figure 33) opens by clicking [Micro Op. Description] item in [ASIP Meister Main Menu] window. With this window, you describe the behavior of each stage of the instruction and interrupt (exception) defined in [Instruction Definition] window. The behavior description consists of declaration of the variable and description of each stage.



ADD #1
ADDU #1
ADDI #1
ADDUI #1
SUB #1
SUBU #1
SUBI #1
SUBUI #1
MULT #1
MULTU #1
DIV #1
DIVU #1
AND #1
ANDI #1
OR #1
ORI #1
XOR #1
XORI #1
SLL #1
SRL #1
SRA #1
SLLI #1
SRLI #1
SRAI #1
SLT #1
SGT #1
SLE #1
SGE #1

Stage	Behavior Description
VARIABLE	
IF	FETCH()
ID	GPR2READ(rs0, rs1)
EXE	ALUEXEC(add, source0, source1)
MEM	
WB	WRITEBACK(rd, result)

Figure 33: [Micro Op. Description] window

"Micro-operation Description Coding Guide" describes the detail of the syntax and statement of the micro behavior operation. Please refer to the document.

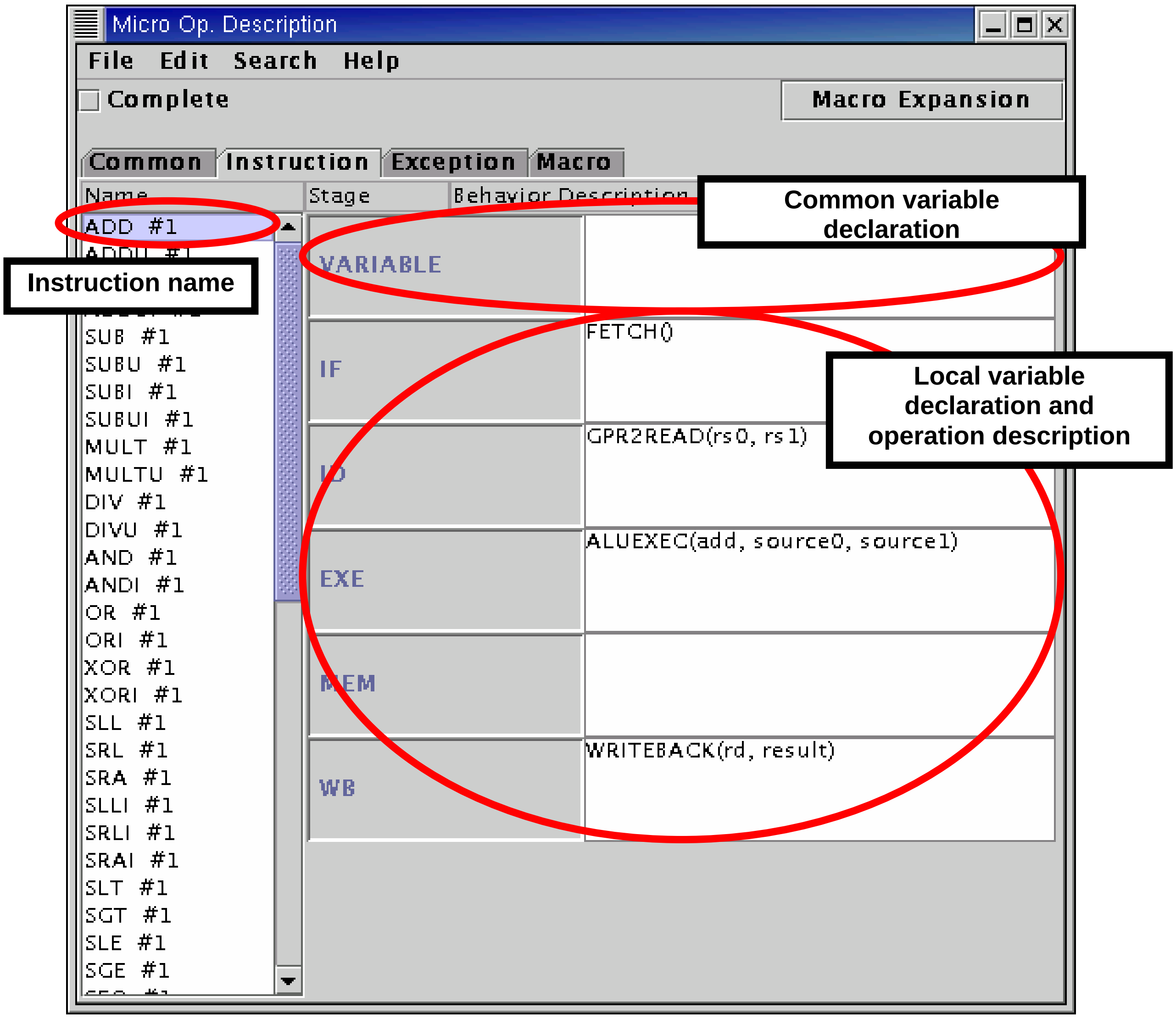
[Micro Op. Description] window has four tabs; [Common], [Instruction], [Exception] and [Macro]. Select [Instruction] to describe the behavior of the instruction, [Exception] to describe the behavior of interrupt and [Macro] to define the macro. The current version does not support [Common].

10.1. Instruction operation definition

To describe the behavior of instructions, please click [Instruction] tab in [Micro Op. Description] window and [Micro Op. Description] window will shows the descriptions of the instructions' behavior (Figure 34).

[Name] column lists the instruction names defined in [Instruction Definition] step. If you select the instruction name from the list, its behavior is displayed in the right side of the window. Initially, the right side of the window is shown blank.

[Stage] column includes [VARIABLE] as a first item and the stage names defined in [Design Goal & Arch Design] step. The variables used in several stages must be defined in [VARIBALE] row and local variables used in one stage can be defined in each stage description. Please describe the behavior of each stage in [Behavior Description] column. You must not use reserved words described in appendix section 4.1 as the variable name.



Stage	Behavior Description
VARIABLE	
IF	FETCH()
ID	GPR2READ(rs0, rs1)
EXE	ALUEXEC(add, source0, source1)
MEM	
WB	WRITEBACK(rd, result)

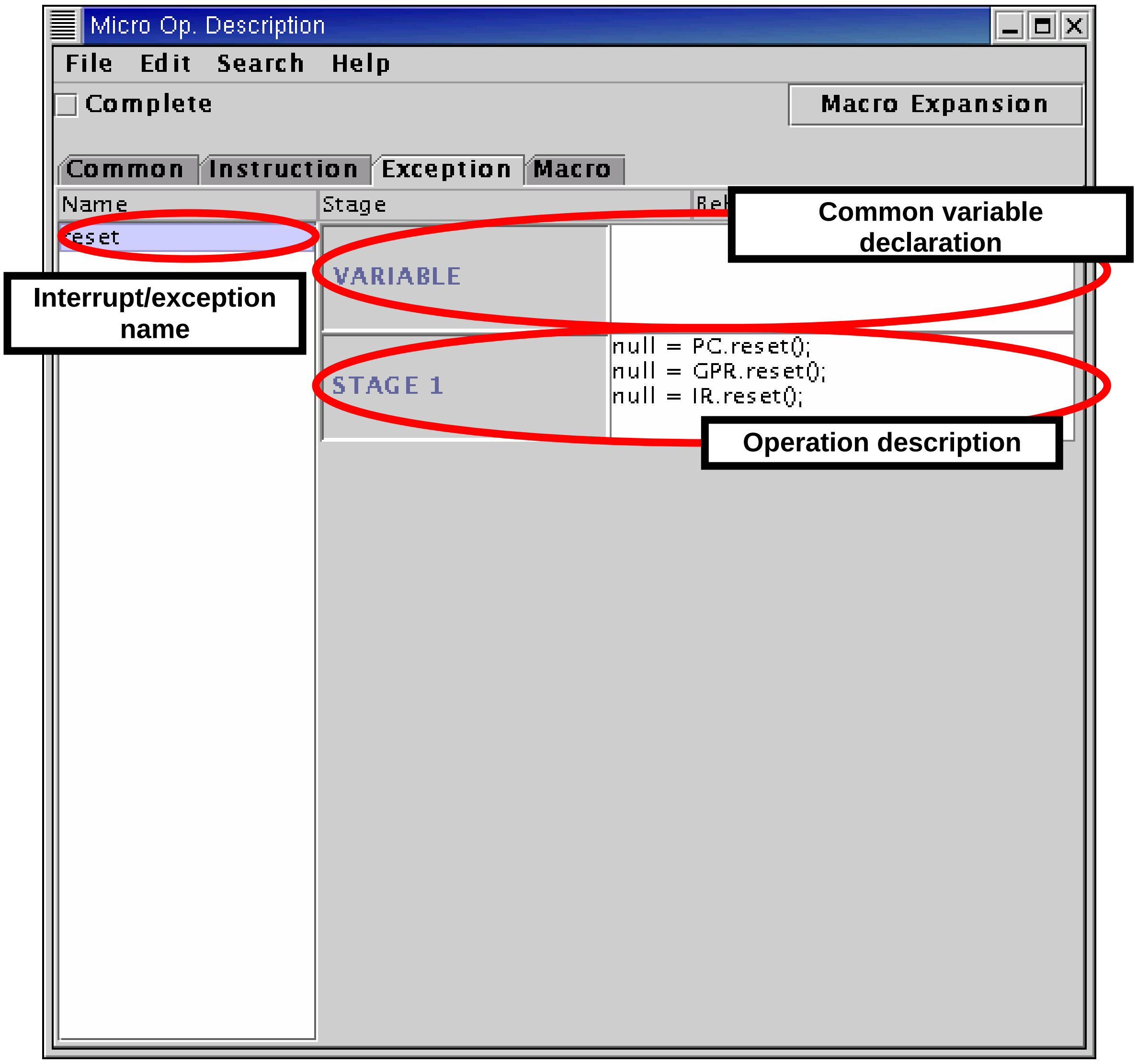
Figure 34: Micro operation description of instructions

10.2. Interrupt operation definition

To describe the behavior of interrupts and exceptions, please click [Exception] tab in [Micro Op. Description] window and the window will show the description of behavior of the interrupts and exceptions (Figure 35).

[Name] column lists the interrupt and exception names defined in [Instruction Definition] step. If you select the interrupt (exception) name from the list, its behavior is displayed in the right side of the window.

[Stage] column includes [VARIABLE] as the first row and [STAGE 1]. Please define variables in [VARIABLE] field and the behavior of the interrupt in [Behavior Description] field of [STAGE 1]. Current version supports only single cycle interrupt operation. You must not use reserved words described in appendix section 4.1 as the variable name.



```
null = PC.reset();
null = GPR.reset();
null = IR.reset();
```

Figure 35: Micro operation description of interrupts and exceptions

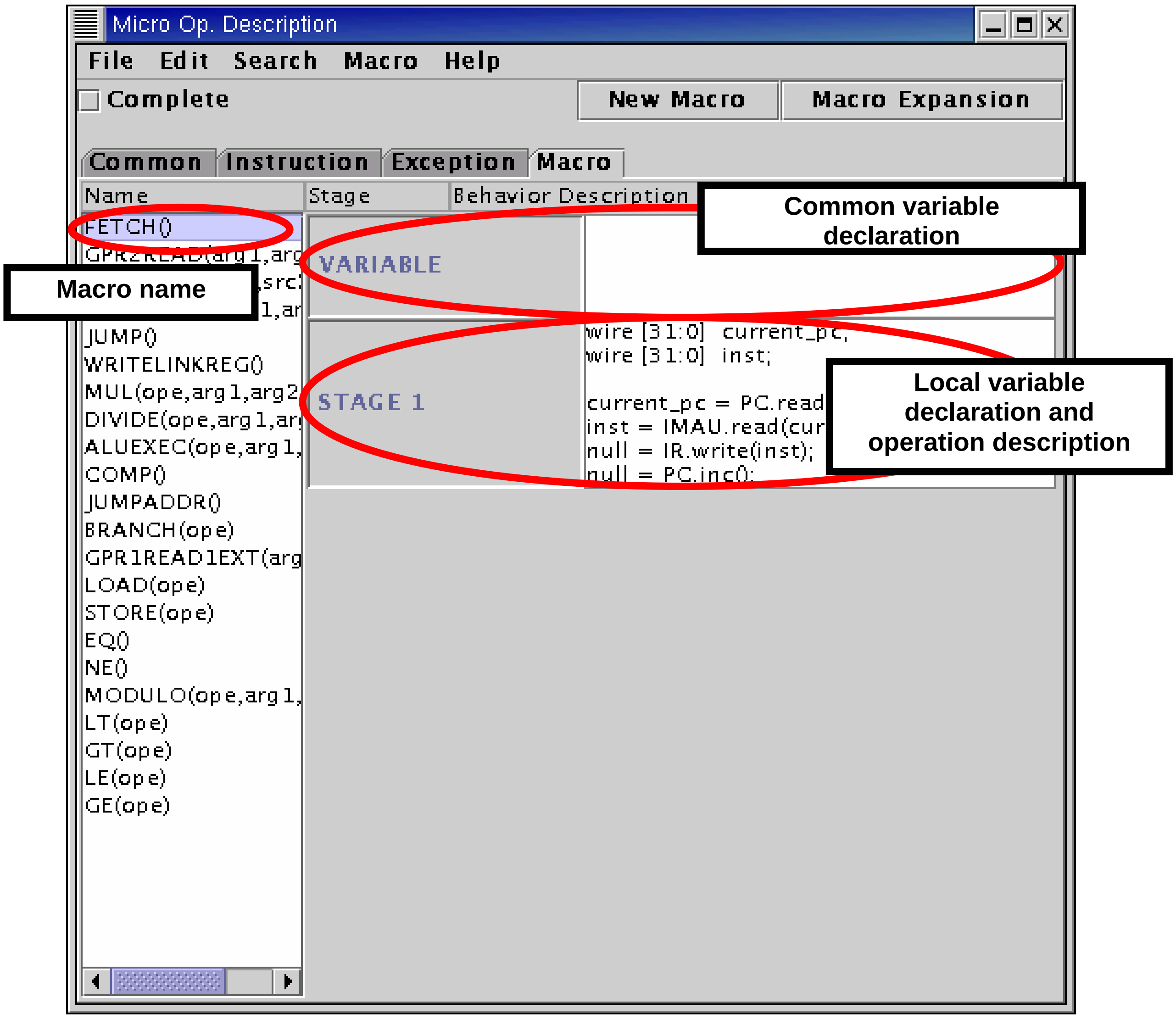
10.3. Macro operation definition

You can simplify the micro behavior description by defining the macro for the behavior that is commonly used among multiple instructions.

To define the macros, please click [Macro] tab in [Micro Op. Description] window and the window will show the definition of macros (Figure 36).

[Name] column lists the macro names. If you select the macro name from the list, its behavior is displayed in the right side of the window.

[Stage] column includes [VARIABLE] as the first row and rows for behavior description. The variables used in several stages must be defined in [VARIBALE] row and local variables used in one stage can be defined in each stage description. Please describe the behavior of each stage in [Behavior Description] column. You must not use reserved words described in appendix section 4.1 as the variable name.

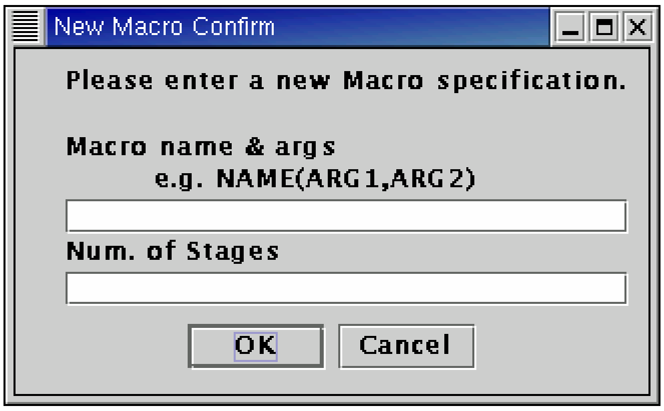


```
FETCH()
GPR2READ(arg1,arg
,src
1,ar
JUMP()
WRITELINKREG()
MUL(ope,arg1,arg2
DIVIDE(ope,arg1,ar
ALUEXEC(ope,arg1,
COMP()
JUMPADDR()
BRANCH(ope)
GPR1READ1EXT(arg
LOAD(ope)
STORE(ope)
EQ()
NE()
MODULO(ope,arg1,
LT(ope)
GT(ope)
LE(ope)
GE(ope)
```

```
wire [31:0]  current_pc;
wire [31:0]  inst;

current_pc = PC.read
inst = IMAU.read(cur
null = IR.write(inst);
null = PC.inc();
```

Figure 36: Definition of macros

To define new macro, please click [New Macro] button in the window and [New Macro Confirm] window will open (Figure 37).



Please enter a new Macro specification.

Macro name & args
e.g. NAME(ARG1,ARG2)

Num. of Stages

Figure 37: [New Macro Confirm] window

In [New Macro Confirm] window, please set the macro name, its arguments and the number of stages. You can define the macro name and arguments like "MACRO(argument1, argument2)". If you use the macro with arguments in the description of the instruction behavior, you can specify the arguments. If you finish the setting, please click [OK] button. After returning to [Micro Op. Description] window, please describe the behavior description for the macro.

10.4. Macro expansion

When the macro is used to describe the behavior of the instruction, interrupt and macro, ASIP Meister has a function to let you check the macro expanded description. To check the macro expanded description, press [Macro Expansion] button. This expands the macro to display its details. Argument strings will be replaced with an actual argument values.

11. HDL Generation

[Generation Confirm] window (Figure 38) opens by clicking [HDL Generation] item in [ASIP Meister Main Menu] window. Use this window to generate the VHDL descriptions of the processor. You can automatically generate the VHDL descriptions of the simulation model used to validate the functionality and the behavior and that of the logical synthesis model. The VHDL description of each resource is described with the abstraction level specified in [Description Style of HDL] settings of [Resource Declaration] step.

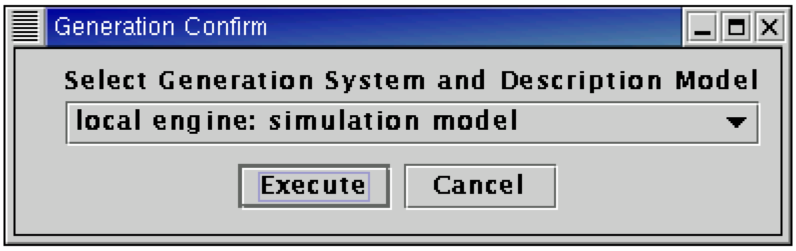
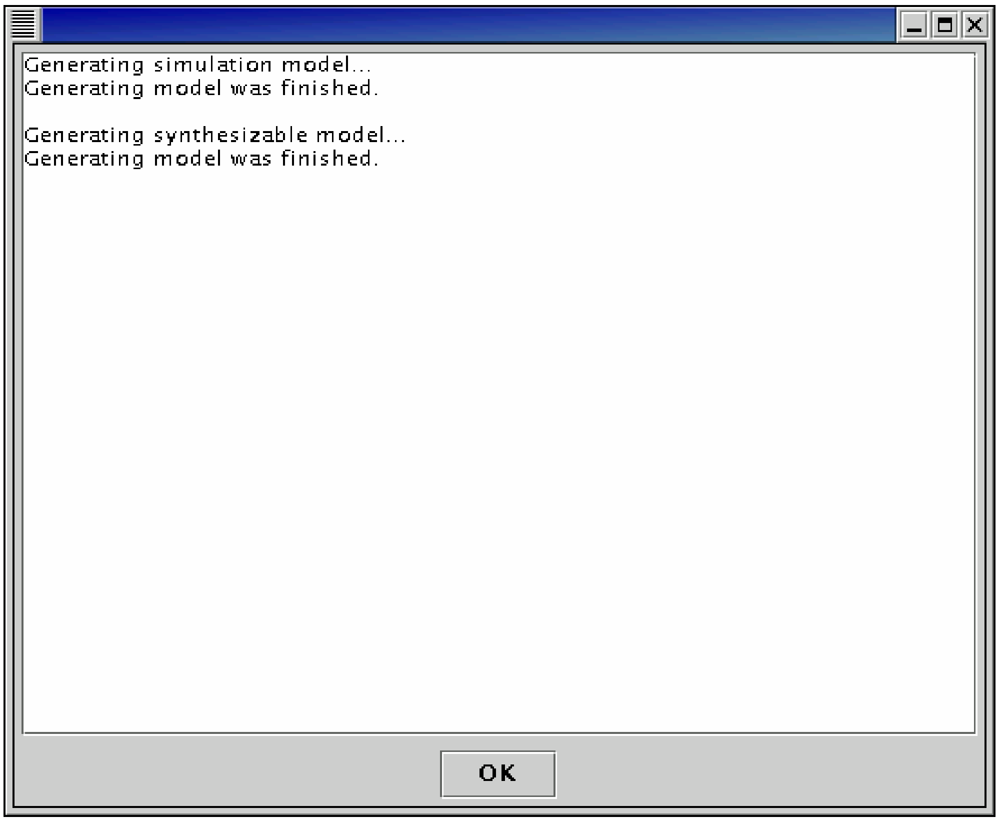

Generation Confirm
Select Generation System and Description Model
local engine: simulation model
Execute
Cancel

Figure 38: [Generation Confirm] window

Please select the generation model from the pull down menu. The details of each engine are as follows;

● [Simulation model]:	Generates the simulation model only.
● [Synthesizable model]:	Generates the logical synthesis model only.
● [sim. model and syn. model]:	Generates both simulation model and logical synthesis model.

After VHDL description generation, the window with the generation completion message (Figure 39) opens. If there are some errors, please check the message and return to fix your settings. Otherwise, click [OK] button.



```
Generating simulation model...
Generating model was finished.

Generating synthesizable model...
Generating model was finished.
```

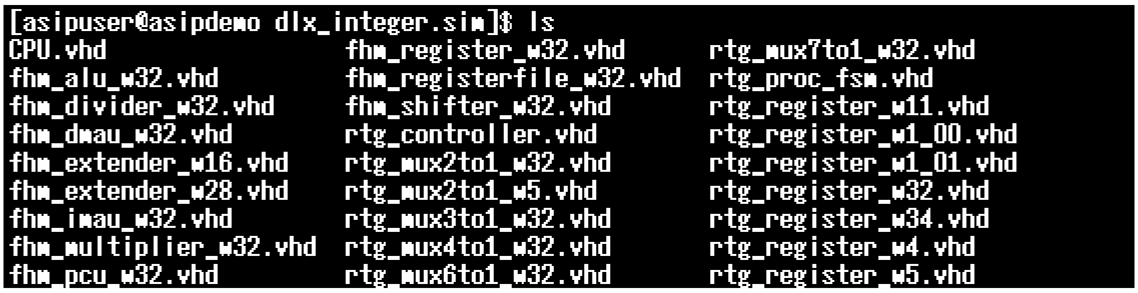
Figure 39: VHDL Generation completion window

When ASIP Meister generates the VHDL descriptions successfully, the directory that contains the descriptions is created under the "meister" directory in the current directory. For example, if you have a design data file named "model.pdb", the description for the simulation model is stored under "meister/model.sim/" directory, and the description for logical synthesis model is stored under "meister/model.syn/" directory.

The VHDL description files has the extension ".vhd". The VHDL file of the top module is named the entity name defined in [Interface definition] with the extension ".vhd".

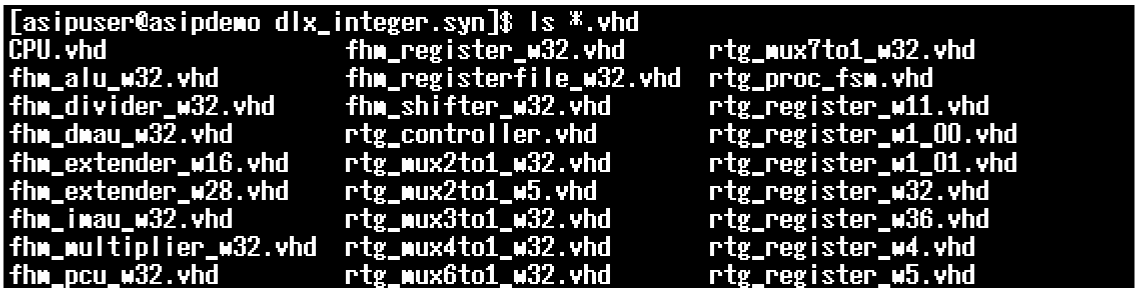
The directory for the synthesizable model also contains the synthesis scripts for Design Compiler® of Synopsys, Inc. The synthesis scripts have the extension ".scr".

Figure 40 to Figure 42 shows the examples of the file generation.



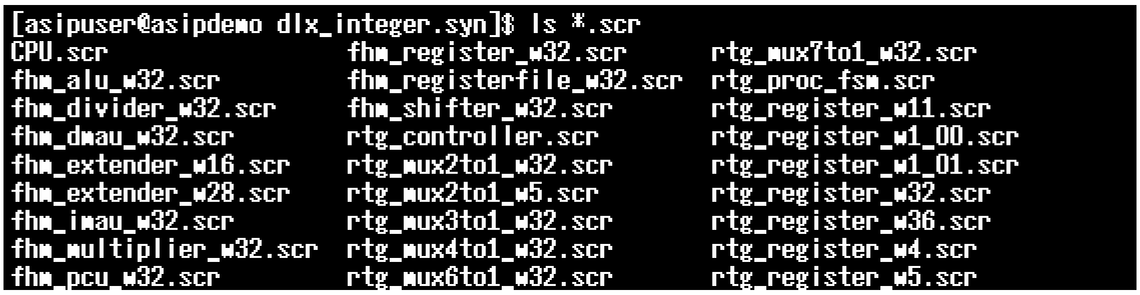
```
[asipuser@asipdemo dlx_integer.sim]$ ls
CPU.vhd                  fhm_register_w32.vhd      rtg_mux7to1_w32.vhd
fhm_alu_w32.vhd          fhm_registerfile_w32.vhd  rtg_proc_fsm.vhd
fhm_divider_w32.vhd      fhm_shifter_w32.vhd       rtg_register_w11.vhd
fhm_dmau_w32.vhd         rtg_controller.vhd        rtg_register_w1_00.vhd
fhm_extender_w16.vhd     rtg_mux2to1_w32.vhd       rtg_register_w1_01.vhd
fhm_extender_w28.vhd     rtg_mux2to1_w5.vhd        rtg_register_w32.vhd
fhm_imau_w32.vhd         rtg_mux3to1_w32.vhd       rtg_register_w34.vhd
fhm_multiplier_w32.vhd   rtg_mux4to1_w32.vhd       rtg_register_w4.vhd
fhm_pcu_w32.vhd          rtg_mux6to1_w32.vhd       rtg_register_w5.vhd
```

Figure 40: Example list of descriptions of the simulation model



```
[asipuser@asipdemo dlx_integer.syn]$ ls *.vhd
CPU.vhd                  fhm_register_w32.vhd      rtg_mux7to1_w32.vhd
fhm_alu_w32.vhd          fhm_registerfile_w32.vhd  rtg_proc_fsm.vhd
fhm_divider_w32.vhd      fhm_shifter_w32.vhd       rtg_register_w11.vhd
fhm_dmau_w32.vhd         rtg_controller.vhd        rtg_register_w1_00.vhd
fhm_extender_w16.vhd     rtg_mux2to1_w32.vhd       rtg_register_w1_01.vhd
fhm_extender_w28.vhd     rtg_mux2to1_w5.vhd        rtg_register_w32.vhd
fhm_imau_w32.vhd         rtg_mux3to1_w32.vhd       rtg_register_w36.vhd
fhm_multiplier_w32.vhd   rtg_mux4to1_w32.vhd       rtg_register_w4.vhd
fhm_pcu_w32.vhd          rtg_mux6to1_w32.vhd       rtg_register_w5.vhd
```

Figure 41: Example list of descriptions of the synthesizable model



```
[asipuser@asipdemo dlx_integer.syn]$ ls *.scr
CPU.scr                  fhm_register_w32.scr      rtg_mux7to1_w32.scr
fhm_alu_w32.scr          fhm_registerfile_w32.scr  rtg_proc_fsm.scr
fhm_divider_w32.scr      fhm_shifter_w32.scr       rtg_register_w11.scr
fhm_dmau_w32.scr         rtg_controller.scr        rtg_register_w1_00.scr
fhm_extender_w16.scr     rtg_mux2to1_w32.scr       rtg_register_w1_01.scr
fhm_extender_w28.scr     rtg_mux2to1_w5.scr        rtg_register_w32.scr
fhm_imau_w32.scr         rtg_mux3to1_w32.scr       rtg_register_w36.scr
fhm_multiplier_w32.scr   rtg_mux4to1_w32.scr       rtg_register_w4.scr
fhm_pcu_w32.scr          rtg_mux6to1_w32.scr       rtg_register_w5.scr
```

Figure 42: Example list of synthesis scripts for the synthesizable model

12. SWdev Generation

[Generation Confirm] window (Figure 43) opens by clicking [SWdev Generation] item in [ASIP Meister Main Menu] window. With this window, you can get the descriptions for the software tools generation engine.

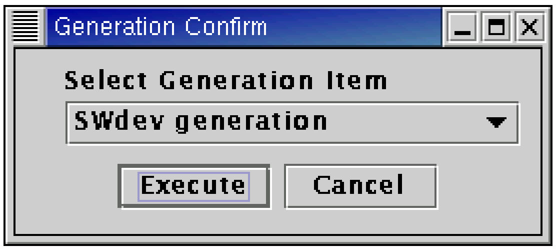


Figure 43: [Generation Confirm] window

In this version of ASIP Meister, only [SWdev generation] menu can be select. So, just click [Execute] button in [Generation Confirm] window.

After architecture description has been generated, the window with the generation completion message (Figure 39) opens. If there are some errors, please check the message and return to fix your settings. Otherwise, click [OK] button.

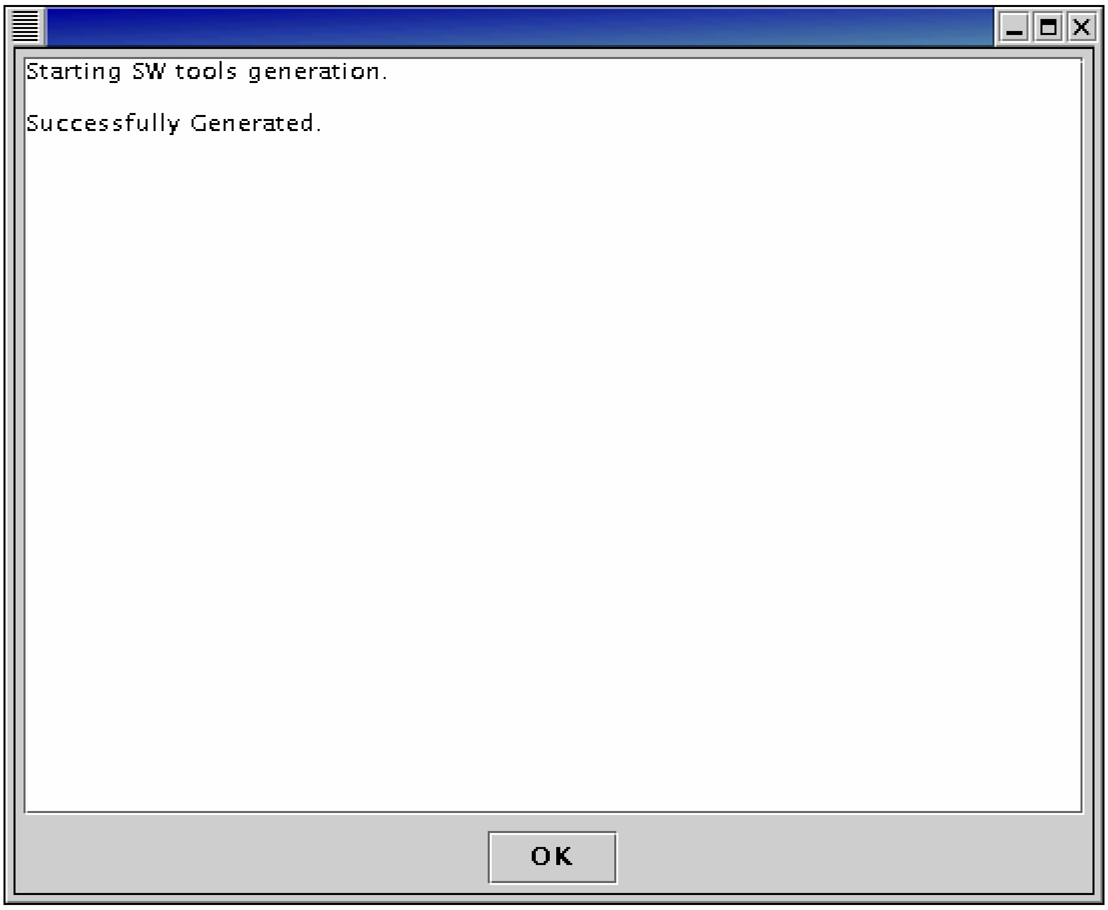
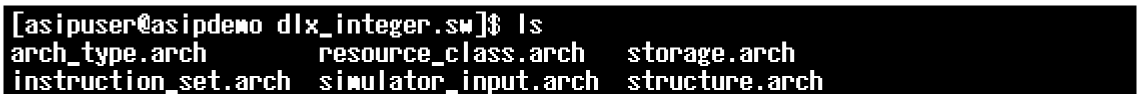

```
Starting SW tools generation.

Successfully Generated.
```

Figure 44: Generation completion window of the description for software tools generation

When ASIP Meister generates the descriptions for the software tools generation engine successfully, the directories that contain the descriptions are created under the “meister” directory in the current directory. For example, if you have a design data file named “model.pdb”, the description for the meta-assembler “meister/model.des” is generated and the descriptions for the compiler generation engine are stored under “meister/model.sw/” directory. Figure 45 show the examples of the file generation.



```
[asipuser@asipdemo dlx_integer.sw]$ ls
arch_type.arch        resource_class.arch   storage.arch
instruction_set.arch  simulator_input.arch  structure.arch
```

Figure 45: Example list of descriptions for the compiler generation engine

III. Appendix

1. Supported Processor models in ASIP Meister

Table 16 shows the available processor model with ASIP Meister.

Table 16: Available processor model with ASIP Meister

Supported design	Unsupported design
Pipeline	Non pipeline
Fixed length instruction	Variable-length instruction
Multi cycle instruction	Multi-length instruction
In-Order completion	Out-of-Order completion
Interlock by structural hazard	Interlock by data hazard
Harvard	Nonharbard
External interruption (single level)	Multiple level external interruption
Internal interruption (exception)	

Please contact to the supporting group for the details of the models not stated above.

2. Registered resources and their parameters

The Table 17 to Table 19 show the parameters for the each resource.

Table 17: Resources in basicfhmdb/computational group

Model	Description	Parameter	Value
alu	ALU	bit_width algorithm	4, 8, 12, 16, 20, 24, 28, 32, 64, 128, 256 rca (ripple carry adder), cla (carry look ahead adder)
adder	Adder	bit_width algorithm	Each value between 1-72, and 128, 256 Rca, cla
divider	Divider	bit_width algorithm adder_algorithm data_type	Every 4 bits value between 4-64 and 128 Seq (sequential), array (array) Rca, cla Unsigned, abs, two_complement
extender	Extender	bit_width bit_width_out	1-63 8, 16, 32, 64
multiplier	Multiplier	bit_widht algorithm adder_algorithm data_type	Every 4 bits value between 4-64, and 128 Seq, array Rca, cla Unsigned, abs, two_complement
barrelshifter	Rotator	bit_width	4, 8, 16, 32, 64, 128
shifter	Arithmetic and logical shifter	bit_width amount	Every 4 bits value between 4-64, and 128 Variable, 1, 2, 4, 8, 16, 32
multiplexer	Multiplexer	bit_widht number_of_ports	1-32, 64, 128 2-16

Table 18: Resources in basicfhmdb/storage group

Model	Description	Parameter	Value
register	Register	bit_width	1-80, 128
registerfile	Register file	bit_width num_register num_read_port num_write_port	Every 4 bits value between 4-64, and 128 4, 16, 32 1, 2, 4 1, 2, 4

Table 19: Resources in workdb/peas group

Model	Description	Parameter	Value
imau	Instruction memory access unit	bit_width	4, 8, 16, 32, 64, 128
dmau	Data memodt access unit	bit_width	4, 8, 16, 32, 64, 128
pcu	Program counter	bit_width increment_step adder_algorithm	4, 8, 16, 32, 64, 128 1, 2, 4, 8 cla, rca
wire_in	Modules for user defined input ports	bit_width	1-80, 128
wire_out	Modules for user defined output ports	bit_width default_output	1-80, 128 fix_to_0, fix_to_1, keep
wire_inout	Modules for user defined input and output ports	bit_width	1-80, 128

3. Limitation in use

ASIP Meister does not support the multi users feature. It is developed for single user only. Therefore, when used by multiple users, each user must install own ASIP Meister on the computer.

4. Reserved words

This section explains reserved words that cannot be used in ASIP Meister design file and VHDL files.

4.1. Reserved words in ASIP Meister design file

The following words cannot be used as data file name, stage name, resource instance name, instruction type name, instruction field name, instruction name, entity name of the target processor, port name, interrupt or exception name, and variable name of micro operation description.

active_value	catch_interrupt	cause_condition	cause_condition_type
clock_port	connect_to	data_memory	decode_error
decode_stage	design_level	direction	dont_care
exec_stage	external_interrupt	fetch_stage	flag_register
for_simulation	for_synthesis	in	inout
instr_memory	instr_register	instr_specific	instr_type
instruction	internal_controller	internal_interrupt	mask_bitpos
mask_condition	mask_register	mask_register	memory_read_stage
memory_write_stage		mod	model null
num_stages	opecode	operand	out
parameter	plain_register	port	program_counter
register_file	register_read_stage	register_write_stage	reserved
reset_interrupt	reset_port	resource	saved_pc
stage	status_register	throw	top_module
wire			

4.2. Reserved words in VHDL file

The following words cannot be used as entity name of the target processor and port name.

abs	access	after	alias	all
and	architecture	array	assert	attribute
begin	block	body	buffer	bus
case	component	configuration	constant	disconnect
downto	else	elsif	end	entity exit
file	for	function	generate	generic
group	guarded	if	impure	in
inertial	inout	is	label	library
linkage	literal	loop	map	mod
nand	new	next	nor	not
null	of	on	open	or
others	out	package	port	postponed
procedure	process	pure	range	record
register	reject	rem	report	return
rol	ror	select	severity	shared
signal	sla	sll	sra	srl
subtype	then	to	transport	type
unaffected	units	until	use	variable
wait	when	while	with	xnor
xor				

IV. Bibliographies

1) M. Itoh, S. Higaki, J. Sato, A. Shiomi, Y. Takeuchi, A. Kitajima, M. Imai, “PEAS-III: An ASIP Design Environment,” in Proc. of ICCD2000, pp.430-436, September 2000.
2) S. Kobayashi, K. Mita, Y. Takeuchi, M. Imai, “A Compiler Generation Method for HW/SW Codesign Based on Configurable Processors,” in IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E85-A, No.12, pp. 2586-2595, December 2002.
3) M. Itoh, Y. Takeuchi, M. Imai, A. Shiomi, “Synthesizable HDL Generation for Pipelined Processors from a Micro-Operation Description,” in IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E83-A, No. 3, pp. 394-400, March 2000.
4) K. Okuda, S. Kobayashi, Y. Takeuchi, M. Imai, “A Simulator Generator Based on Configurable VLIW Model Considering Synthesizable HW Description and SW Tools Generation,” in Proc. of SASIMI 2003, pp. 152-159, April 2003.
5) T. Morifuji, Y. Takeuchi, J. Sato, M. Imai, “Flexible Hardware Model: Implementation and Effectiveness,” in Proc. of SASIMI ’97, pp.83-89, December 1997.
6) T. Sasaki, S. Kobayashi, T. Maeda, M. Itoh, Y. Takeuchi, M. Imai, “Rapid Prototyping of Complex Instructions for Embedded Processors using PEAS-III,” in Proc. of SASIMI 2001, pp 61-66, Nara, Japan, October 2001.
7) Y. Kobayashi, S. Kobayashi, K. Okuda, K. Sakanushi, Y. Takeuchi, M. Imai, “Synthesizable HDL Generation Method for Configurable VLIW Processors,” in Proc. of ASP-DAC 2004, pp.843-846, January 2004.
8) A. Kitajima, M. Itoh, J. Sato, A. Shiomi, Y. Takeuchi, M. Imai, “Effectiveness of the ASIP Design System PEAS-III in Design of Pipelined Processors,” in Proc. of ASP-DAC 2001, pp.649-654, February 2001.
9) S. Kobayashi, K. Mita, Y. Takeuchi, M. Imai, “Rapid Prototyping of JPEG Encoder using the ASIP Development System: PEAS-III,” in Proc. of ICASSP 2003, Vol. 2, pp. 485-488, April 2003.
10) S. Kobayashi, K. Mita, Y. Takeuchi, M. Imai, “Design Space Exploration for DSP Applications using the ASIP Development System PEAS-III,” in Proc. of ICASSP 2002, Vol. 3, pp. 3168-3171, May 2002.
11) J. Sato, A. Y. Alomary, Y. Honma, T. Nakata, A. Shiomi, N. Hikichi, M. Imai, “PEAS-I: A Hardware/Software Co-design System for ASIP Development,” in IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E77-A, No. 3, pp. 483-491, March 1994.
12) N. N. Binh, M. Imai, Y. Takeuchi, “An Optimization Algorithm for High Performance ASIP Design with Considering the RAM and ROM Sizes,” in IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E81-A, No.12, pp. 2612-2620, December 1998.

Index