Workshop on Essential Abstractions in GCC

Spim Machine Descriptions

GCC Resource Center
(www.cse.iitb.ac.in/grc)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

2 July 2012

Outline

- Systematic construction of machine descriptions
- Retargetting GCC to spim
    - spim is mips simulator developed by James Larus
    - RISC machine
    - Assembly level simulator: No need of assembler, linkers, or libraries
- Level 0 of spim machine descriptions
- Level 1 of spim machine descriptions

Outline

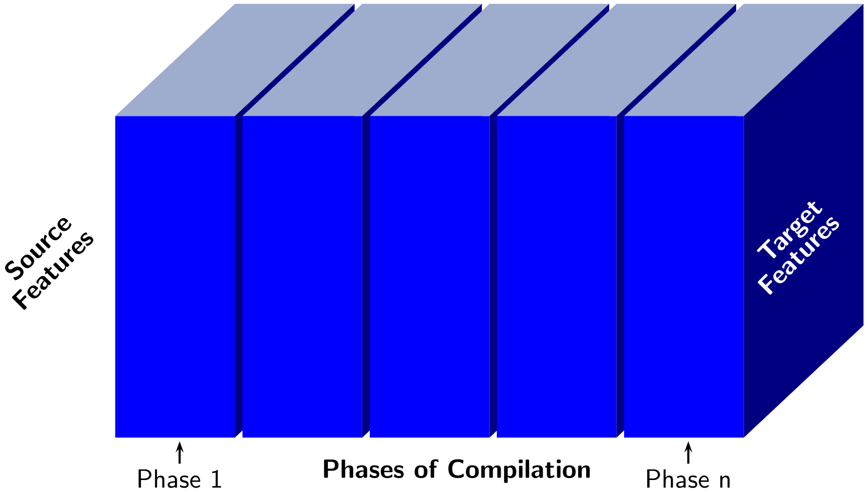
Notes

Part 1

Systematic Construction of Machine Descriptions

Notes

In Search of Modularity in Retargetable Compilation



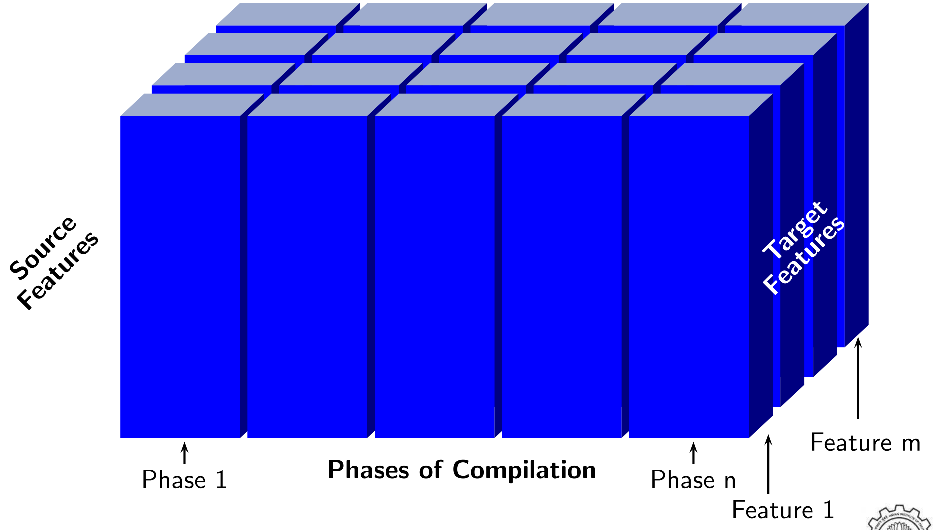
2 July 2012 Spim MD: Systematic Construction of Machine Descriptions 2/61

In Search of Modularity in Retargetable Compilation

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

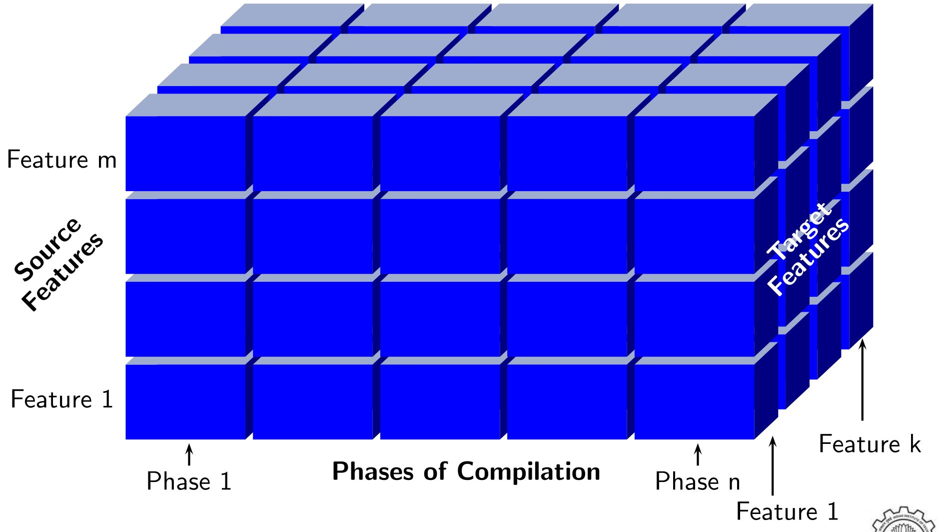
In Search of Modularity in Retargetable Compilation



In Search of Modularity in Retargetable Compilation

Notes

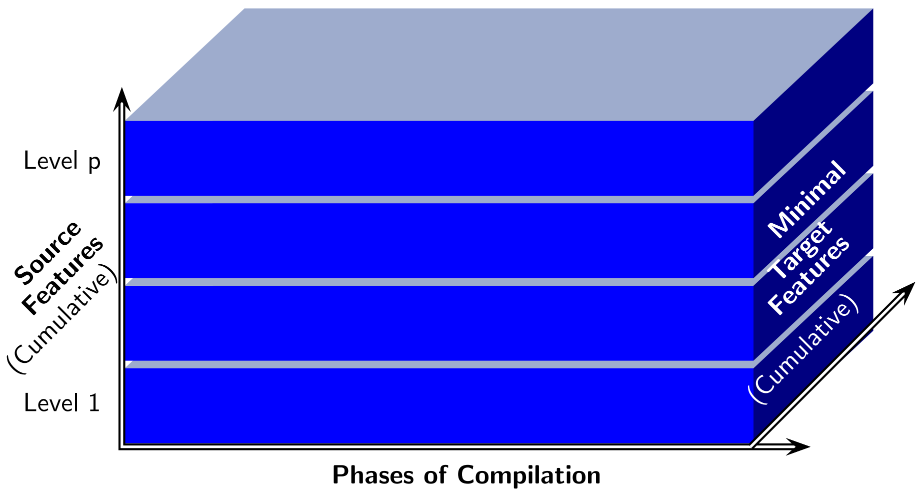
In Search of Modularity in Retargetable Compilation



In Search of Modularity in Retargetable Compilation

Notes

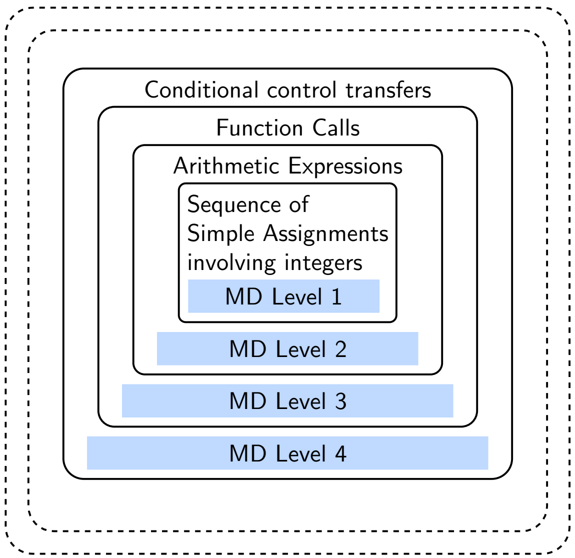
In Search of Modularity in Retargetable Compilation



In Search of Modularity in Retargetable Compilation

Notes

Systematic Development of Machine Descriptions



Systematic Development of Machine Descriptions

Notes

Systematic Development of Machine Descriptions

- Define different levels of source language
- Identify the minimal information required in the machine description to support each level
  - Successful compilation of any program, and
  - correct execution of the generated assembly program.
- Interesting observations
  - It is the increment in the source language which results in understandable increments in machine descriptions rather than the increment in the target architecture.
  - If the levels are identified properly, the increments in machine descriptions are monotonic.

Systematic Development of Machine Descriptions

Notes

Part 2

Retargeting GCC to Spim: A Recap

Notes

Retargeting GCC to Spim

- Registering spim target with GCC build process
- Making machine description files available
- Building the compiler

Retargeting GCC to Spim

Notes

Registering Spim with GCC Build Process

We want to add multiple descriptions:

- Step 1. In the file `$(SOURCE_D)/config.sub`
  Add to the `case $basic_machine`
  - `spim*` in the part following
    `# Recognize the basic CPU types without company name.`
  - `spim*-*` in the part following
    `# Recognize the basic CPU types with company name.`

Registering Spim with GCC Build Process

Notes

Registering Spim with GCC Build Process

- Step 2a. In the file $(SOURCE_D)/gcc/config.gcc

  In case ${target} used for defining cpu_type, i.e. after the line

  ```
  # Set default cpu_type, tm_file, tm_p_file and xm_file ...
  ```

  add the following case

  ```
  spim*-*-*)
       cpu_type=spim
       ;;
  ```

  This says that the machine description files are available in the directory $(SOURCE_D)/gcc/config/spim.

2 July 2012 Spim MD: Retargeting GCC to Spim: A Recap 10/61

Registering Spim with GCC Build Process

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Registering Spim with GCC Build Process

- Step 2b. In the file $(SOURCE_D)/gcc/config.gcc

  Add the following in the case ${target} for
  # Support site-specific machine types.

  ```
  spim*-*-*)
        gas=no
        gnu_ld=no
        file_base="`echo ${target} | sed 's/-.*$//'`"
        tm_file="${cpu_type}/${file_base}.h"
        md_file="${cpu_type}/${file_base}.md"
        out_file="${cpu_type}/${file_base}.c"
        tm_p_file="${cpu_type}/${file_base}-protos.h"
        echo ${target}
          ;;
  ```


2 July 2012 Spim MD: Retargeting GCC to Spim: A Recap 11/61

Registering Spim with GCC Build Process

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Building a Cross-Compiler for Spim

- Normal cross compiler build process attempts to use the generated cc1 to compile the emulation libraries (LIBGCC) into executables using the assembler, linker, and archiver.
- We are interested in only the cc1 compiler.
  Add a new target in the Makefile.in

```
.PHONY: cc1
cc1:
    make all-gcc TARGET-gcc=cc1$(exeext)
```


2 July 2012 Spim MD: Retargeting GCC to Spim: A Recap 12/61

Building a Cross-Compiler for Spim

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Building a Cross-Compiler for Spim

- Create directories ${BUILD_D} and in a tree not rooted at ${SOURCE_D}.
- Change the directory to ${BUILD_D} and execute the commands

```
$ cd ${BUILD_D}
$ ${SOURCE_D}/configure --target=spim<n>
$ make cc1
```

- Pray for 10 minutes :-)

2 July 2012 Spim MD: Retargeting GCC to Spim: A Recap 13/61

Building a Cross-Compiler for Spim

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Part 3

Level 0 of Spim Machine Descriptions

Notes

2 July 2012 Spim MD: Level 0 of Spim Machine Descriptions 14/61

Sub-levels of Level 0

Three sub-levels

- Level 0.0: Merely build GCC for spim simulator
  Does not compile any program (i.e. compilation aborts)
- Level 0.1: Compiles empty void functions

```
void fun(int p1, int p2)
{
     int v1, v2;
}
```

```
void fun()
{
  L: goto L;
}
```

- Level 0.2: Incorporates complete activation record structure
  Required for Level 1

Sub-levels of Level 0

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Category of Macros in Level 0

Category	Level 0.0	Level 0.1	Level 0.2
Memory Layout	complete	complete	complete
Registers	partial	partial	complete
Addressing Modes	none	partial	partial
Activation Record Conventions	dummy	dummy	complete
Calling Conventions	dummy	dummy	partial
Assembly Output Format	dummy	partial	partial

- Complete specification of activation record in level 0.2 is not necessary but is provided to facilitate local variables in level 1.
- Complete specification of registers in level 0.2 follows the complete specification of activation record.

2 July 2012 Spim MD: Level 0 of Spim Machine Descriptions 15/61

Category of Macros in Level 0

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Memory Layout Related Macros for Level 0

```
#define BITS_BIG_ENDIAN 0
#define BYTES_BIG_ENDIAN 0
#define WORDS_BIG_ENDIAN 0
#define UNITS_PER_WORD 4
#define PARM_BOUNDARY 32
#define STACK_BOUNDARY 64
#define FUNCTION_BOUNDARY 32
```

```
#define BIGGEST_ALIGNMENT 64
#define STRICT_ALIGNMENT 0
#define MOVE_MAX 4
#define Pmode SImode
#define FUNCTION_MODE SImode
#define SLOW_BYTE_ACCESS 0
#define CASE_VECTOR_MODE SImode
```

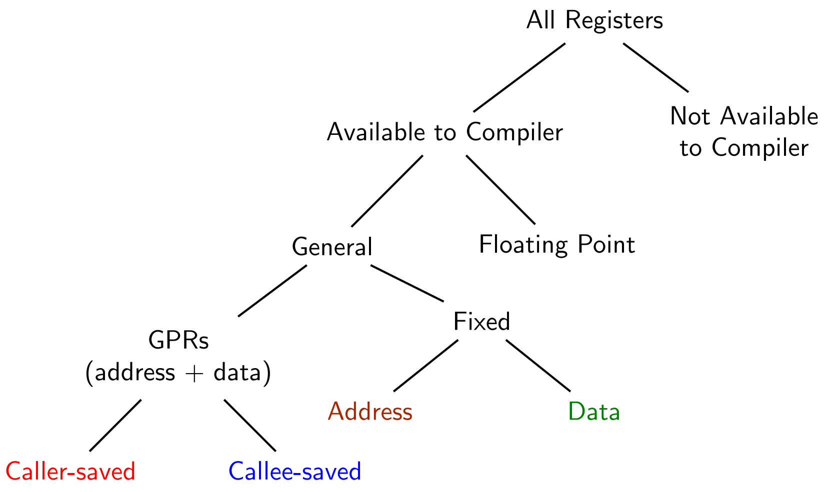

2 July 2012 Spim MD: Level 0 of Spim Machine Descriptions 16/61

Memory Layout Related Macros for Level 0

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Register Categories for Spim



2 July 2012 Spim MD: Level 0 of Spim Machine Descriptions 17/61

Register Categories for Spim

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Registers in Spim

`$zero`	00	32		constant data
`$at`	01	32		NA
`$v0`	02	32,64	result	caller
`$v1`	03	32	result	caller
`$a0`	04	32,64	argument	caller
`$a1`	05	32	argument	caller
`$a2`	06	32,64	argument	caller
`$a3`	07	32	argument	caller
`$t0`	08	32,64	temporary	caller
`$t1`	09	32	temporary	caller
`$t2`	10	32,64	temporary	caller
`$t3`	11	32	temporary	caller
`$t4`	12	32,64	temporary	caller
`$t5`	13	32	temporary	caller
`$t6`	14	32,64	temporary	caller
`$t7`	15	32	temporary	caller

`$s0`	16	32,64	temporary	callee
`$s1`	17	32	temporary	callee
`$s2`	18	32,64	result	callee
`$s3`	19	32	result	callee
`$s4`	20	32,64	temporary	callee
`$s5`	21	32	temporary	callee
`$s6`	22	32,64	temporary	callee
`$s7`	23	32	temporary	callee
`$t8`	24	32,64	temporary	caller
`$t9`	25	32	temporary	caller
`$k0`	26	32,64		NA
`$k1`	27	32		NA
`$gp`	28	32,64	global pointer	address
`$sp`	29	32	stack pointer	address
`$fp`	30	32,64	frame pointer	address
`$ra`	31	32	return address	address

2 July 2012 Spim MD: Level 0 of Spim Machine Descriptions 18/61

Registers in Spim

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Register Information in Level 0.2

$zero,$at

$k0,$k1

$gp,$sp,$fp,$ra

```
#define FIRST_PSEUDO_REGISTER 32

#define FIXED_REGISTERS \
/* not for global */ \
/* register allocation */ \
{  1,1,0,0, 0,0,0,0, \
   0,0,0,0, 0,0,0,0, \
   0,0,0,0, 0,0,0,0, \
   0,0,1,1, 1,1,1,1}

#define CALL_USED_REGISTERS \
/* Caller-saved registers */ \
{  1,1,1,1, 1,1,1,1, \
   1,1,1,1, 1,1,1,1, \
   0,0,0,0, 0,0,0,0, \
   1,1,1,1, 1,1,1,1 }
```

```
/* Register sizes */

#define HARD_REGNO_NREGS(R,M)\
((GET_MODE_SIZE (M) + \
       UNITS_PER_WORD - 1) \
       / UNITS_PER_WORD)

#define HARD_REGNO_MODE_OK(R,M)\
      hard_regno_mode_ok (R, M)

#define MODES_TIEABLE_P(M1,M2)\
      modes_tieable_p (M1,M2)
```

$s0 to $s7

Register Information in Level 0.2

Notes

Register Classes in Level 0.2

```
enum reg_class \
{  NO_REGS, CALLER_SAVED_REGS, \
   CALLEE_SAVED_REGS, BASE_REGS, \
   GENERAL_REGS, ALL_REGS, \
   LIM_REG_CLASSES \
};
#define N_REG_CLASSES \
       LIM_REG_CLASSES
#define REG_CLASS_NAMES \
{  "NO_REGS","CALLER_SAVED_REGS",\
   "CALLEE_SAVED_REGS", \
   "BASE_REGS", "GEN_REGS", \
   "ALL_REGS" \
}
#define REG_CLASS_CONTENTS \
/* Register numbers */ \
{  0x00000000,0xff00ffff, \
   0x00ff0000,0xf0000000, \
   0x0cfffff3,0xffffffff }
```

address registers

```
#define REGNO_REG_CLASS(REGNO) \
    regno_reg_class(REGNO)
#define BASE_REG_CLASS \
    BASE_REGS
#define INDEX_REG_CLASS NO_REGS
#define REG_CLASS_FROM_LETTER(c)\
     NO_REGS
#define REGNO_OK_FOR_BASE_P(R) 1
#define REGNO_OK_FOR_INDEX_P(R) 0
#define PREFERRED_RELOAD_CLASS(X,C)\
    CLASS
/* Max reg required for a class */
#define CLASS_MAX_NREGS(C, M) \
     ((GET_MODE_SIZE (M) + \
           UNITS_PER_WORD - 1) \
           / UNITS_PER_WORD)
#define LEGITIMATE_CONSTANT_P(x) \
    legitimate_constant_p(x)
```


Register Classes in Level 0.2

Notes

function calling conventions

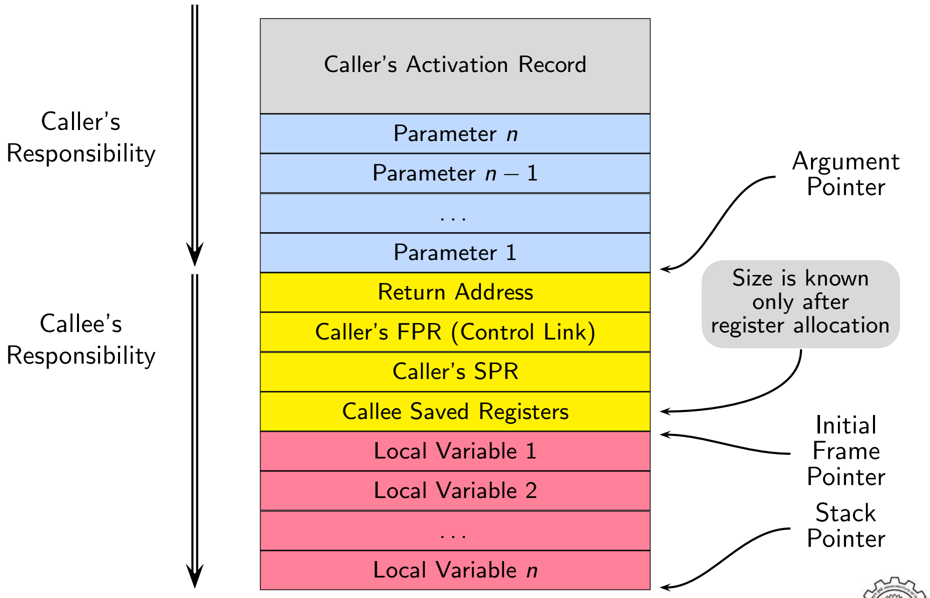
pass arguments on stack. return values goes in register $v0 (in level 1).

```
#define RETURN_POPS_ARGS(FUN, TYPE, SIZE) 0
#define FUNCTION_ARG(CUM, MODE, TYPE, NAMED)  0
#define FUNCTION_ARG_REGNO_P(r) 0
    /*Data structure to record the information about args passed in
     *registers. Irrelevant in this level so a simple int will  do. */
#define CUMULATIVE_ARGS int
#define INIT_CUMULATIVE_ARGS(CUM, FNTYPE, LIBNAME, FNDECL, NAMED_ARGS) \
   { CUM = 0; }
#define FUNCTION_ARG_ADVANCE(cum, mode, type, named) cum++
#define FUNCTION_VALUE(valtype, func) function_value()
#define FUNCTION_VALUE_REGNO_P(REGN) ((REGN) == 2)
```


function calling conventions

Notes

Activation Record Structure in Spim



Activation Record Structure in Spim

Notes

Minimizing Registers for Accessing Activation Records

Reduce four pointer registers (stack, frame, args, and hard frame) to fewer registers.

```
#define ELIMINABLE_REGS
{{FRAME_POINTER_REGNUM,      STACK_POINTER_REGNUM},
 {FRAME_POINTER_REGNUM,      HARD_FRAME_POINTER_REGNUM},
 {ARG_POINTER_REGNUM,        STACK_POINTER_REGNUM},
 {HARD_FRAME_POINTER_REGNUM, STACK_POINTER_REGNUM}
}


/Recomputes new offsets, after eliminating./

#define INITIAL_ELIMINATION_OFFSET(FROM, TO, VAR)
        (VAR) = initial_elimination_offset(FROM, TO)
```


2 July 2012 Spim MD: Level 0 of Spim Machine Descriptions 23/61

Minimizing Registers for Accessing Activation Records

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Specifying Activation Record

```
#define STARTING_FRAME_OFFSET  starting_frame_offset ()

#define FIRST_PARM_OFFSET(FUN) 0

#define STACK_POINTER_REGNUM 29

#define FRAME_POINTER_REGNUM 1

#define HARD_FRAME_POINTER_REGNUM 30

#define ARG_POINTER_REGNUM HARD_FRAME_POINTER_REGNUM

#define FRAME_POINTER_REQUIRED 0
```


2 July 2012 Spim MD: Level 0 of Spim Machine Descriptions 24/61

Specifying Activation Record

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Level 0.0 Machine Description File

Empty :-)

Level 0.0 Machine Description File

Notes

Operations in Level 0

Operations	Level 0.0	Level 0.1	Level 0.2
JUMP direct	dummy	actual	actual
JUMP indirect	dummy	dummy	dummy
NOP	dummy	actual	actual
MOV	not required	partial	partial
RETURN	not required	partial	partial

spim0.0.c

```
rtx gen_jump (...)
{ return 0; }
rtx gen_indirect_jump (...)
{ return 0; }
rtx gen_nop ()
{ return 0; }
```

spim0.0.h

```
#define CODE_FOR_indirect_jump 8
```

spim0.2.md

```
(define_insn "jump"
   [(set (pc)
     (label_ref (match_operand 0 "" ""))
    )]
   ""
   "j %l0"
)
```


Operations in Level 0

Notes

Operations in Level 0

Operations	Level 0.0	Level 0.1	Level 0.2
JUMP direct	dummy	actual	actual
JUMP indirect	dummy	dummy	dummy
NOP	dummy	actual	actual
MOV	not required	partial	partial
RETURN	not required	partial	partial

Only define_expand. No define_insn.

```
(define_expand "movsi"
     [(set (match_operand:SI 0 "nonimmediate_operand" "")
         (match_operand:SI 1 "general_operand" ""))
     )]
     ""
     {
         if(GET_CODE(operands[0])==MEM && GET_CODE(operands[1])!=REG)
             {
                 if(can_create_pseudo_p())
                 {
                         operands[1]=force_reg(SImode,operands[1]);
                 }}}
)
```

spim0.2.md

2 July 2012 Spim MD: Level 0 of Spim Machine Descriptions 26/61

Operations in Level 0

Notes

2 July 2012 Spim MD: Level 0 of Spim Machine Descriptions 26/61

Operations in Level 0

Operations	Level 0.0	Level 0.1	Level 0.2
JUMP direct	dummy	actual	actual
JUMP indirect	dummy	dummy	dummy
NOP	dummy	actual	actual
MOV	not required	partial	partial
RETURN	not required	partial	partial

spim0.2.md

```
(define_expand "epilogue"
   [(clobber (const_int 0))]
   ""
   { spim_epilogue();
     DONE;
   }
)
(define_insn "IITB_return"
   [(return)]
   ""
   "jr \\$ra"
)
```

Only return.
No epilogue code.

spim0.2.c

```
void spim_epilogue()
{
    emit_insn(gen_IITB_return());
}
```


2 July 2012 Spim MD: Level 0 of Spim Machine Descriptions 26/61

Operations in Level 0

Notes

Operations in Level 0

Operations	Level 0.0	Level 0.1	Level 0.2
JUMP direct	dummy	actual	actual
JUMP indirect	dummy	dummy	dummy
NOP	dummy	actual	actual
MOV	not required	partial	partial
RETURN	not required	partial	partial

```
(define_insn "nop"
   [(const_int 0)]
   ""
   "nop"
)
```


2 July 2012 Spim MD: Level 0 of Spim Machine Descriptions 26/61

Operations in Level 0

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Part 4

Level 1 of Spim Machine Descriptions

Notes

Increments for Level 1

- Addition to the source language
  - Assignment statements involving integer constant, integer local or global variables.
  - Returning values. (No calls, though!)
- Changes in machine descriptions
  - Minor changes in macros required for level 0
    `$zero` now belongs to new class Assembly output needs to change
  - Some function bodies expanded
  - New operations included in the `.md` file

`diff -w` shows the changes!

2 July 2012 Spim MD: Level 1 of Spim Machine Descriptions 27/61

Increments for Level 1

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Operations Required in Level 1

Operation	Primitive Variants	Implementation	Remark
$Dest \leftarrow Src$	$R_i \leftarrow R_j$	`move rj, ri`	
	$R \leftarrow M_{global}$	`lw r, m`	
	$R \leftarrow M_{local}$	`lw r, c($fp)`	
	$R \leftarrow C$	`li r, c`	
	$M \leftarrow R$	`sw r, m`	
RETURN Src	RETURN Src	`$v0` $\leftarrow$ Src `j $ra`	 level 0
$Dest \leftarrow Src_1 + Src_2$	$R_i \leftarrow R_j + R_k$	`add ri, rj, rk`	
	$R_i \leftarrow R_j + C$	`addi ri, rj, c`	

2 July 2012 Spim MD: Level 1 of Spim Machine Descriptions 28/61

Operations Required in Level 1

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Move Operations in spim1.md

- Multiple primitive variants require us to map a single operation in IR to multiple RTL patterns
  ⇒ use define_expand
- Ensure that the second operand is in a register

```
(define_expand "movsi"
  [(set (match_operand:SI 0 "nonimmediate_operand" "")
        (match_operand:SI 1 "general_operand" "")
  )]
  ""
  { if(GET_CODE(operands[0])==MEM &&
       GET_CODE(operands[1])!=REG &&
        (can_create_pseudo_p()) /* force conversion only */
                              /* before register allocation */
    { operands[1]=force_reg(SImode,operands[1]); }
  }
)
```


2 July 2012 Spim MD: Level 1 of Spim Machine Descriptions 29/61

Move Operations in spim1.md

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Move Operations in spim1 Compiler for Assignment a = b

```
(define_expand "movsi"
  [(set (match_operand:SI 0 "nonimmediate_operand" "")
        (match_operand:SI 1 "general_operand" "")
  )]
  ""
  { if(GET_CODE(operands[0])==MEM &&
       GET_CODE(operands[1])!=REG &&
        (can_create_pseudo_p()) /* force conversion only */
                              /* before register allocation */
    { operands[1]=force_reg(SImode,operands[1]); }
  }
)

(insn 6 5 7 3 t.c:25 (set (reg:SI 38)
        (mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars)
                (const_int -4 [0xfffffffc])) [0 b+0 S4 A32])) -1 (nil))
(insn 7 6 8 3 t.c:25 (set (mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars)
                (const_int -8 [0xfffffff8])) [0 a+0 S4 A32])
        (reg:SI 38)) -1 (nil))
```


2 July 2012 Spim MD: Level 1 of Spim Machine Descriptions 30/61

Move Operations in spim1 Compiler for Assignment a = b

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Move Operations in spim1.md

- Load from Memory $R \leftarrow M$

```
(define_insn "*load_word"
   [(set (match_operand:SI 0 "register_operand" "=r")
         (match_operand:SI 1 "memory_operand" "m"))]
   ""
   "lw \t%0, %m1"
)
```

- Load Constant $R \leftarrow C$

```
(define_insn "*constant_load"
   [(set (match_operand:SI 0 "register_operand" "=r")
         (match_operand:SI 1 "const_int_operand" "i"))]
   ""
   "li \t%0, %c1"
```


2 July 2012 Spim MD: Level 1 of Spim Machine Descriptions 31/61

Move Operations in spim1.md

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Move Operations in spim1.md

- Register Move $R_i \leftarrow R_j$

```
(define_insn "*move_regs"
   [(set (match_operand:SI 0 "register_operand" "=r")
         (match_operand:SI 1 "register_operand" "r")
   )]
   ""
   "move \t%0,%1"
)
```

- Store into $M \leftarrow R$

```
(define_insn "*store_word"
   [(set (match_operand:SI 0 "memory_operand" "=m")
         (match_operand:SI 1 "register_operand" "r"))]
   ""
   "sw \t%1, %m0"
)
```


2 July 2012 Spim MD: Level 1 of Spim Machine Descriptions 32/61

Move Operations in spim1.md

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Code Generation in `spim1` Compiler for Assignment `a = b`

- RTL statements

```
(insn 6 5 7 3 t.c:25 (set (reg:SI 38)
        (mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars)
                (const_int -4 [0xfffffffc])) [0 b+0 S4 A32])) -1 (nil))
(insn 7 6 8 3 t.c:25 (set (mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-
                (const_int -8 [0xfffffff8])) [0 a+0 S4 A32])
        (reg:SI 38)) -1 (nil))
```

- Generated Code

```
lw $v0, -16($fp)
sw $v0, -20($fp)
```


2 July 2012 Spim MD: Level 1 of Spim Machine Descriptions 33/61
Code Generation in spim1 Compiler for Assignment a = b

Notes

Using register `$zero` for constant 0

- Introduce new register class `zero_register_operand` in `spim1.h` and define `move_zero`

```
(define_insn "IITB_move_zero"
   [(set (match_operand:SI 0 "nonimmediate_operand" "=r,m")
         (match_operand:SI 1 "zero_register_operand" "z,z")
   )]
   ""
   "@
   move \t%0,%1
   sw \t%1, %m0"
)
```

- How do we get `zero_register_operand` in an RTL?

2 July 2012 Spim MD: Level 1 of Spim Machine Descriptions 34/61
Using register $zero for constant 0

Notes

Using register `$zero` for constant 0

- Use `define_expand "movsi"` to get `zero_register_operand` in an RTL

```
if(GET_CODE(operands[1])==CONST_INT && INTVAL(operands[1])==0)
{
    emit_insn(gen_IITB_move_zero(operands[0],
                                 gen_rtx_REG(SImode,0)));
    DONE;
}
else /* Usual processing */
```

- `DONE` says do not generate the RTL template associated with `"movsi"`
- required template is generated by `emit_insn(gen_IITB_move_zero(...))`

Using register `$zero` for constant 0

Notes

Supporting Addition in Level 1

```
(define_insn "addsi3"
    [(set (match_operand:SI 0 "register_operand" "=r,r")
          (plus:SI (match_operand:SI 1 "register_operand" "r,r")
                   (match_operand:SI 2 "nonmemory_operand" "r,i"))
     )]
    ""
    "@
     add \t%0, %1, %2
     addi \t%0, %1, %c2"
)
```

- Constraints combination 1 of three operands: R, R, R
- Constraints combination 2 of three operands: R, R, C

Supporting Addition in Level 1

Notes

Comparing `movsi` and `addsi3`

- `movsi` uses `define_expand` whereas `addsi3` uses combination of operands
- Why not use constraints for `movsi` too?
- Combination of operands is used during pattern matching and not during expansion
  - We will need to support memory as both source and destination
  - Will also allow memory to memory move in RTL
    We will not know until assembly emission which one is a load instruction and which one is a store instruction

2 July 2012 Spim MD: Level 1 of Spim Machine Descriptions 37/61

Comparing movsi and addsi3

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Part 5

Conclusions

Notes

Conclusions

- Incremental construction of machine description files is very instructive
- Increments in machine descriptions is governed by increments in source language
- Machine characteristics need to be specified in C macros and C functions
  - Does not seem amenable to incremental construction
  - Seems difficult to a novice
- Specifying instructions seems simpler and more systematic
  - Is amenable to incremental construction
  - The concept of minimal machine descriptions is very useful
- `define_insn` and `define_expand` are the main constructs used on machine descriptions

Conclusions

Notes

Part 6

Constructs Supported in Level 2

Notes

Arithmetic Operations Required in Level 2

Operation	Primitive Variants	Implementation	Remark
$Dest \leftarrow Src_1 - Src_2$	$R_i \leftarrow R_j - R_k$	`sub ri, rj, rk`	level 2
$Dest \leftarrow -Src$	$R_i \leftarrow -R_j$	`neg ri, rj`	
$Dest \leftarrow Src_1/Src_2$	$R_i \leftarrow R_j/R_k$	`div rj, rk` `mflo ri`	
$Dest \leftarrow Src_1\%Src_2$	$R_i \leftarrow R_j\%R_k$	`rem ri, rj, rk`	
$Dest \leftarrow Src_1 * Src_2$	$R_i \leftarrow R_j * R_k$	`mul ri, rj, rk`	

Arithmetic Operations Required in Level 2

Notes

Bitwise Operations Required in Level 2

Operation	Primitive Variants	Implementation	Remark
$Dest \leftarrow Src_1 \ll Src_2$	$R_i \leftarrow R_j \ll R_k$	`sllv ri, rj, rk`	level 2
	$R_i \leftarrow R_j \ll C_5$	`sll ri, rj, c`	
$Dest \leftarrow Src_1 \gg Src_2$	$R_i \leftarrow R_j \gg R_k$	`srav ri, rj, rk`	
	$R_i \leftarrow R_j \gg C_5$	`sra ri, rj, c`	
$Dest \leftarrow Src_1 \& Src_2$	$R_i \leftarrow R_j \& R_k$	`and ri, rj, rk`	
	$R_i \leftarrow R_j \& C$	`andi ri, rj, c`	
$Dest \leftarrow Src_1 \mid Src_2$	$R_i \leftarrow R_j \mid R_k$	`or ri, rj, rk`	
	$R_i \leftarrow R_j \mid C$	`ori ri, rj, c`	
$Dest \leftarrow Src_1 \hat{} Src_2$	$R_i \leftarrow R_j \hat{} R_k$	`xor ri, rj, rk`	
	$R_i \leftarrow R_j \hat{} C$	`xori ri, rj, c`	
$Dest \leftarrow \sim Src$	$R_i \leftarrow \sim R_j$	`not ri, rj`	

Bitwise Operations Required in Level 2

Notes

Divide Operation in `spim2.md` using define_insn

- For division, the spim architecture imposes use of multiple asm instructions for single operation.
- Two ASM instructions are emitted using single RTL pattern

```
(define_insn "divsi3"
 [(set (match_operand:SI 0 "register_operand" "=r")
       (div:SI (match_operand:SI 1 "register_operand" "r")
               (match_operand:SI 2 "register_operand" "r"))
 )]
 ""
 "div\\t%1, %2\\n\\tmflo\\t%0"
)
```


2 July 2012 Spim MD: Constructs Supported in Level 2 41/61

Divide Operation in spim2.md using define_insn

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Advantages/Disadvantages of using define_insn

- Very simple to add the pattern
- Primitive target feature represented as single insn pattern in `.md`
- Unnecessary atomic grouping of instructions
- May hamper optimizations in general, and instruction scheduling, in particluar

2 July 2012 Spim MD: Constructs Supported in Level 2 42/61

Advantages/Disadvantages of using define_insn

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Divide Operation in spim2.md using define_expand

- The RTL pattern can be expanded into two different RTLs.

```
(define_expand "divsi3"
 [(parallel[(set (match_operand:SI 0 "register_operand" "")
      (div:SI (match_operand:SI 1 "register_operand" "")
              (match_operand:SI 2 "register_operand" ""))
  )
  (clobber (reg:SI 26))
  (clobber (reg:SI 27))])]
  ""
  {
    emit_insn(gen_IITB_divide(gen_rtx_REG(SImode,26),
                              operands[1], operands[2]));
    emit_insn(gen_IITB_move_from_lo(operands[0],
                              gen_rtx_REG(SImode,26)));
    DONE;
  }
)
```


2 July 2012 Spim MD: Constructs Supported in Level 2 43/61

Divide Operation in spim2.md using define_expand

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Divide Operation in spim2.md using define_expand

- Divide pattern equivalent to div instruction in architecture.

```
(define_insn "IITB_divide"
 [(parallel[(set (match_operand:SI 0 "LO_register_operand" "=q")
      (div:SI (match_operand:SI 1 "register_operand" "r")
              (match_operand:SI 2 "register_operand" "r"))
 )
 (clobber (reg:SI 27))])]
 ""
 "div t%1, %2"
)
```


2 July 2012 Spim MD: Constructs Supported in Level 2 44/61

Divide Operation in spim2.md using define_expand

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Divide Operation in spim2.md using define_expand

- Moving contents of special purpose register LO to/from general purpose register

```
(define_insn "IITB_move_from_lo"
 [(set (match_operand:SI 0 "register_operand" "=r")
       (match_operand:SI 1 "LO_register_operand" "q"))]
 ""
 "mflo \\t%0"
)

(define_insn "IITB_move_to_lo"
 [(set (match_operand:SI 0 "LO_register_operand" "=q")
       (match_operand:SI 1 "register_operand" "r"))]
 ""
 "mtlo \\t%1"
)
```


Divide Operation in spim2.md using define_expand

Notes

Divide Operation in spim2.md using define_expand

- Divide pattern equivalent to div instruction in architecture.

```
(define_insn "modsi3"
 [(parallel[(set (match_operand:SI 0 "register_operand" "=r")
       (mod:SI (match_operand:SI 1 "register_operand" "r")
               (match_operand:SI 2 "register_operand" "r"))
 )
 (clobber (reg:SI 26))
 (clobber (reg:SI 27))])]
 ""
 "rem \t%0, %1, %2"
)
```


Divide Operation in spim2.md using define_expand

Notes

Advantages/Disadvantages of Using `define_expand` for Division

- Two instructions are seperated out at GIMPLE to RTL conversion phase
- Both instructions can undergo all RTL optimizations independently
- C interface is needed in md
- Compilation becomes slower and requires more space

2 July 2012 Spim MD: Constructs Supported in Level 2 47/61

Advantages/Disadvantages of Using `define_expand` for Division

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Divide Operation in `spim2.md` using define_split

```
(define_split
 [(parallel [(set (match_operand:SI 0 "register_operand" "")
    (div:SI (match_operand:SI 1 "register_operand" "")
       (match_operand:SI 2 "register_operand" ""))
    )
    (clobber (reg:SI 26))
    (clobber (reg:SI 27))])]
    ""
 [(parallel [(set (match_dup 3)
    (div:SI (match_dup 1)
       (match_dup 2)))
    (clobber (reg:SI 27))])
    (set (match_dup 0)
      (match_dup 3))
 ]
        "operands[3]=gen_rtx_REG(SImode,26); "
)
```

```
[(parallel[
  (set (match_operand:SI 0 "LO_register_operand" "=q")
   (div:SI (match_operand:SI 1 "register_operand" "r")
    (match_operand:SI 2 "register_operand" "r")))
  (clobber (reg:SI 27))])]
```

```
[(set (match_operand:SI 0 "register_operand" "=r")
  (match_operand:SI 1 "LO_register_operand" "q"))]
```


2 July 2012 Spim MD: Constructs Supported in Level 2 48/61

Divide Operation in `spim2.md` using define_split

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Part 7

Constructs Supported in Level 3

Notes

Operations Required in Level 3

Operation	Primitive Variants	Implementation	Remark

$Dest \leftarrow fun(P_1, \ldots, P_n)$	call L_{fun}, n	`lw` r_i, `[SP+c1]`
		`sw` r_i, `[SP]`
		:
		`lw` r_i, `[SP+c2]`
		`sw` r_i, `[SP-n*4]`
			Level 1

		jal L	New
		$\mathtt{Dest} \leftarrow \$v0$	level 1

$fun(P_1, P_2, \ldots, P_n)$	call L_{fun}, n	`lw` r_i, `[SP+c1]`
		`sw` r_i, `[SP]`
		:
		`lw` r_i, `[SP+c2]`
		`sw` r_i, `[SP-n*4]`
			Level 1

		jal L	New

2 July 2012 Spim MD: Constructs Supported in Level 3 49/61

Operations Required in Level 3

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Call Operation in spim3.md

```
(define_insn "call"
 [(call (match_operand:SI 0 "memory_operand" "=m")
        (match_operand:SI 1 "immediate_operand" "i"))
  (clobber (reg:SI 31))
 ]
 ""
 "*
   return emit_asm_call(operands,0);
 "
)
```


2 July 2012 Spim MD: Constructs Supported in Level 3 50/61

Call Operation in spim3.md

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Call Operation in spim3.md

```
(define_insn "call_value"
 [(set (match_operand:SI 0 "register_operand" "=r")
       (call (match_operand:SI 1 "memory_operand" "m")
             (match_operand:SI 2 "immediate_operand" "i")))
  (clobber (reg:SI 31))
 ]
 ""
 "*
   return emit_asm_call(operands,1);
 "
)
```


2 July 2012 Spim MD: Constructs Supported in Level 3 51/61

Call Operation in spim3.md

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Activation Record Generation during `Call`

- Operations performed by caller
  - Push parameters on stack.
  - Load return address in return address register.
  - Transfer control to Callee.
- Operations performed by callee
  - Push Return address stored by caller on stack.
  - Push caller's Frame Pointer Register.
  - Push caller's Stack Pointer.
  - Save callee saved registers, if used by callee.
  - Create local variables frame.
  - Start callee body execution.

Caller's Activation Record
Parameter n
Parameter $n-1$
...
Parameter 1
Return Address
Caller's FPR (Control Link)
Caller's SPR
Callee Saved Registers
Local Variable 1
Local Variable 2
...
Local Variable n

Activation Record Generation during `Call`

Notes

Prologue in `spim3.md`

```
(define_expand "prologue"
 [(clobber (const_int 0))]
 ""
 {
    spim_prologue();
    DONE;
 })
```

```
(set (mem:SI (reg:SI $sp))
     (reg:SI 31 $ra))

(set (mem:SI (plus:SI (reg:SI $sp)
                      (const_int -4 )))
     (reg:SI $sp))

(set (mem:SI (plus:SI (reg:SI $sp)
                      (const_int -8 )))
     (reg:SI $fp))

(set (reg:SI $fp)
     (reg:SI $sp))

(set (reg:SI $sp)
        (plus:SI (reg:SI $fp)
            (const_int -36)))
```


Prologue in `spim3.md`

Notes

Epilogue in `spim3.md`

```
(define_expand "epilogue"
 [(clobber (const_int 0))]
 ""

    spim_epilogue();
    DONE;
 )
```

```
(set (reg:SI $sp)
     (reg:SI $fp))

(set (reg:SI $fp)
     (mem:SI (plus:SI (reg:SI $sp)
        (const_int -8 ))))

(set (reg:SI $ra)
(mem:SI (reg:SI $sp)))

(parallel [
    (return)
    (use (reg:SI $ra))])
```


2 July 2012 Spim MD: Constructs Supported in Level 3 54/61

Epilogue in spim3.md

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Part 8

Constructs Supported in Level 4

Notes

Operations Required in Level 4

Operation	Primitive Variants	Implementation	Remark
$Src_1 < Src_2$? goto L : PC	$CC \leftarrow R_i < R_j$ CC $<$ 0 ? goto L : PC	blt r_i, r_j,L	
$Src_1 > Src_2$? goto L : PC	$CC \leftarrow R_i > R_j$ CC $>$ 0 ? goto L : PC	bgt r_i, r_j,L	
$Src_1 \leq Src_2$? goto L : PC	$CC \leftarrow R_i \leq R_j$ CC $\leq$ 0 ? goto L : PC	ble r_i, r_j,L	
$Src_1 \geq Src_2$? goto L : PC	$CC \leftarrow R_i \geq R_j$ CC $\geq$ 0 ? goto L : PC	bge r_i, r_j,L	

2 July 2012 Spim MD: Constructs Supported in Level 4 55/61

Operations Required in Level 4

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Operations Required in Level 4

Operation	Primitive Variants	Implementation	Remark
$Src_1 == Src_2$? goto L : PC	$CC \leftarrow R_i == R_j$ CC $==$ 0 ? goto L : PC	beq r_i, r_j,L	
$Src_1 \neq Src_2$? goto L : PC	$CC \leftarrow R_i \neq R_j$ CC $\neq$ 0 ? goto L : PC	bne r_i, r_j,L	

2 July 2012 Spim MD: Constructs Supported in Level 4 56/61

Operations Required in Level 4

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Conditional Branch Instruction in spim4.md

```
(define_insn "cbranchsi4"
    [(set (pc)
        (if_then_else
            (match_operator:SI 0 "comparison_operator"
            [(match_operand:SI 1 "register_operand" "")
            (match_operand:SI 2 "register_operand" "")])
                (label_ref (match_operand 3 "" ""))
            (pc)))]
    ""
    "*
        return conditional_insn(GET_CODE(operands[0]),operands);
    "
)
```


2 July 2012 Spim MD: Constructs Supported in Level 4 57/61

Conditional Branch Instruction in spim4.md

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Support for Branch pattern in spim4.c

```
char *
conditional_insn (enum rtx_code code,rtx operands[])
{
    switch(code)
    {
         case EQ:return "beq %1, %2, %l3";
         case NE:return "bne %1, %2, %l3";
         case GE:return "bge %1, %2, %l3";
         case GT:return "bgt %1, %2, %l3";
         case LT:return "blt %1, %2, %l3";
         case LE:return "ble %1, %2, %l3";
         case GEU:return "bgeu %1, %2, %l3";
         case GTU:return "bgtu %1, %2, %l3";
         case LTU:return "bltu %1, %2, %l3";
         case LEU:return "bleu %1, %2, %l3";
         default: /* Error. Issue ICE */
    }
}
```


2 July 2012 Spim MD: Constructs Supported in Level 4 58/61

Support for Branch pattern in spim4.c

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Alternative for Branch: Conditional compare in spim4.md

```
(define_code_iterator cond_code
     [lt ltu eq ge geu gt gtu le leu ne])

(define_expand "cmpsi"
 [(set (cc0) (compare
             (match_operand:SI 0 "register_operand" "")
             (match_operand:SI 1 "nonmemory_operand" "")))]
 ""
 {
     compare_op0=operands[0];
     compare_op1=operands[1];
     DONE;
 }
)
```


2 July 2012 Spim MD: Constructs Supported in Level 4 59/61

Alternative for Branch: Conditional compare in spim4.md

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Alternative for Branch: Branch pattern in spim4.md

```
(define_expand "b<code>"
 [(set (pc) (if_then_else (cond_code:SI (match_dup 1)
           (match_dup 2))
       (label_ref (match_operand 0 "" ""))
       (pc)))]
 ""
 {
    operands[1]=compare_op0;
    operands[2]=compare_op1;
    if(immediate_operand(operands[2],SImode))
    {
       operands[2]=force_reg(SImode,operands[2]);
    }
 }
)
```


2 July 2012 Spim MD: Constructs Supported in Level 4 60/61

Alternative for Branch: Branch pattern in spim4.md

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Alternative for Branch: Branch pattern in `spim4.md`

```
(define_insn "*insn_b<code>"
  [(set (pc)
        (if_then_else
   (cond_code:SI
      (match_operand:SI 1 "register_operand" "r")
              (match_operand:SI 2 "register_operand" "r"))
           (label_ref (match_operand 0 "" ""))
           (pc)))]
        ""
        "*
                return conditional_insn(<CODE>,operands);
        "
)
```


2 July 2012 Spim MD: Constructs Supported in Level 4 61/61

Alternative for Branch: Branch pattern in spim4.md

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay