*Workshop on Essential Abstractions in GCC*

# Incremental Machine Descriptions for Spim: Levels 0 and 1

GCC Resource Center

(www.cse.iitb.ac.in/grc)

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

July 2009

*Part 1*

## *Retargeting GCC to Spim: A Recap*



## Outline

- 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



## Retargeting GCC to Spim

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



July 09 Spim MD Levels 0 & 1: Retargeting GCC to Spim: A Recap 2/32

## Retargeting GCC to Spim

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay



## Registering Spim with GCC Build Process

We want to add multiple descriptions:

- Step 1. In the file `$(SOURCE)/config.sub`
  Add to the case `$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.`



July 09 Spim MD Levels 0 & 1: Retargeting GCC to Spim: A Recap 3/32

## Registering Spim with GCC Build Process

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay

## Registering Spim with GCC Build Process

- Step 2. In the file `$(SOURCE)/gcc/config.gcc`
  - In case `${target}` used for defining cpu_type, add
    ```
    spim*-*-*)
        cpu_type=spim
        ;;
    ```
    This specifies the directory `$(SOURCE)/gcc/config/spim` in which the machine descriptions files are supposed to be made available.
  - In case `${target}` for
    `# Support site-specific machine types.`
    add
    ```
    spim*-*-*)
        gas=no
        gnu_ld=no
        tm_file=spim/$target_noncanonical.h
        md_file=spim/$target_noncanonical.md
        out_file=spim/$target_noncanonical.c
        tm_p_file=spim/$target_noncanonical-protos.h
        ;;
    ```





Registering Spim with GCC Build Process

Notes





## 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.





Building a Cross-Compiler for Spim

Notes

## Building a Cross-Compiler for Spim

- Create directories `$(BUILD)` and `$(INSTALL)` in a tree not rooted at `$(SOURCE)`.
- Change the directory to `$(BUILD)` and execute the command

```
$(SOURCE)/configure --prefix=$(INSTALL)
  --target=spim<n> --enable-languages=c
```

files `spim<n>.h`, `spim<n>.md`, `spim<n>.c` in the directory `$(SOURCE)/gcc/config/spim`

- `make cc1`
- Pray for 5 minutes :-)



July 09 Spim MD Levels 0 & 1: Retargeting GCC to Spim: A Recap 6/32

Building a Cross-Compiler for Spim

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay

*Part 2*

# *Level 0 of Spim Machine Descriptions*

## 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 parameterless void function

```
void fun()
{
}
```

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

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



Notes

July 09 Spim MD Levels 0 & 1: Level 0 of Spim Machine Descriptions 7/32

Sub-levels of Level 0

Notes

Essential Abstrations 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.



Notes

July 09 Spim MD Levels 0 & 1: Level 0 of Spim Machine Descriptions 8/32

Category of Macros in Level 0

Notes

Essential Abstrations 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
```





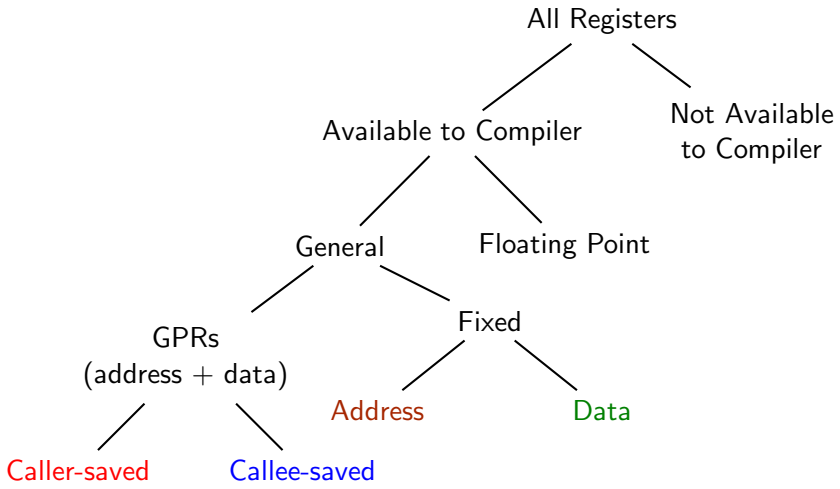
## Memory Layout Related Macros for Level 0

Notes





## Register Categories for Spim








## Register Categories for Spim

Notes



## 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 |





## Registers in Spim

Notes





## 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 */ \
{  0x00000000,0x0200ff00, \
   0x00ff0000,0xf2fffffc, \
   0xffffffff,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)
```



July 09 Spim MD Levels 0 & 1: Level 0 of Spim Machine Descriptions 13/32

## Register Classes in Level 0.2

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay



## Addressing Modes

```
/ Validate use of labels as symbolic references or numeric addresses ×
/
#define CONSTANT_ADDRESS_P(X) constant_address_p(X)
/ Since we don't have base indexed mode, we do not need more than one
  register for any address. /
#define MAX_REGS_PER_ADDRESS 1
/ Validate the addressing mode of an operand of an insn /
#define GO_IF_LEGITIMATE_ADDRESS(mode,x,label) ...
...
#define LEGITIMIZE_ADDRESS(x,oldx,mode,win)
  rtx IITB_rtx_op;
            IITB_rtx_op=legitimize_address(x,oldx,mode);
    if(memory_address_p(mode,IITB_rtx_op))
      x=IITB_rtx_op;
                  goto win;

#define GO_IF_MODE_DEPENDENT_ADDRESS(addr,label)
```

Address of data in the program being compiled

Control transfer in the compiler source



July 09 Spim MD Levels 0 & 1: Level 0 of Spim Machine Descriptions 14/32

## Addressing Modes

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay

## Function Calling Conventions

Pass arguments on stack. Return values goes in register $v0 (in level 1).

```
#define ACCUMULATE_OUTGOING_ARGS  0
                              /* Callee does not pop args */
#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 LIBCALL_VALUE(MODE) function_value()
#define FUNCTION_VALUE_REGNO_P(REGN) ((REGN) == 2)
```





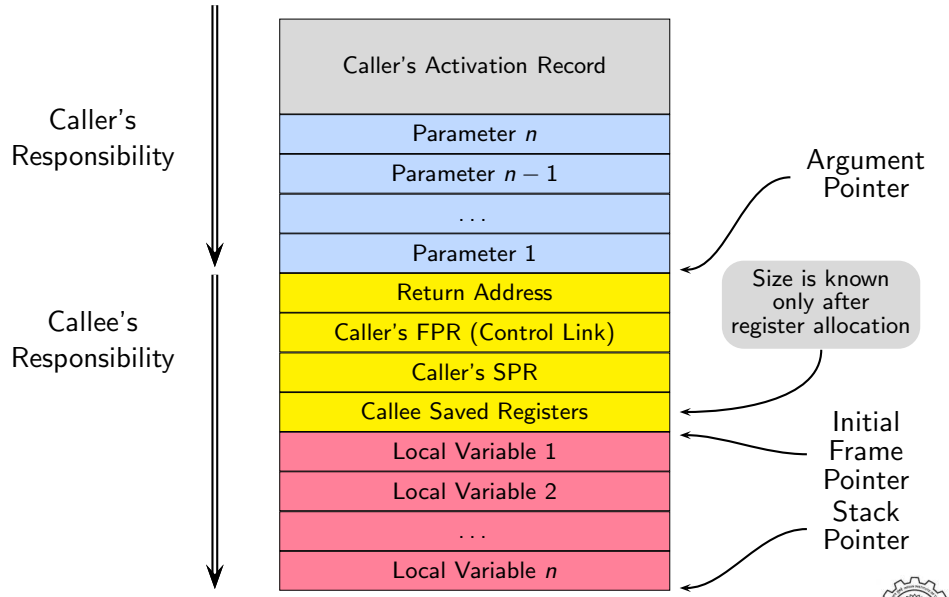
Function Calling Conventions

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}
}

#define CAN_ELIMINATE(FROM, TO)
((FROM == FRAME_POINTER_REGNUM &&
  (TO == STACK_POINTER_REGNUM || TO == HARD_FRAME_POINTER_REGNUM))
  || (FROM == ARG_POINTER_REGNUM && TO == STACK_POINTER_REGNUM)
  || (FROM == HARD_FRAME_POINTER_REGNUM && TO == STACK_POINTER_REGNUM))

/Recomputes new offsets, after eliminating./

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



July 09 Spim MD Levels 0 & 1: Level 0 of Spim Machine Descriptions 17/32

Minimizing Registers for Accessing Activation Records

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay



## Specifying Activation Record

```
#define STACK_GROWS_DOWNWARD 1

#define FRAME_GROWS_DOWNWARD 1

#define STARTING_FRAME_OFFSET
   starting_frame_offset ()

#define STACK_POINTER_OFFSET 0

#define FIRST_PARM_OFFSET(FUN) 0

#define STACK_POINTER_REGNUM 29
```

```
/* "Initial" frame pointer */
/* before register allocation */
#define FRAME_POINTER_REGNUM 1

/* "Final" frame pointer */
/* after register allocation */
#define HARD_FRAME_POINTER_REGNUM 30

#define ARG_POINTER_REGNUM
   HARD_FRAME_POINTER_REGNUM

#define FRAME_POINTER_REQUIRED 0
```



July 09 Spim MD Levels 0 & 1: Level 0 of Spim Machine Descriptions 18/32

Specifying Activation Record

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay

## Operations in Level 0.0

- In principle `spim0.0.md` can be empty.
- However, this results in empty data structures in the C source. Empty arrays in declarations are not acceptable in ANSI C.
- If we remove `-pedantic` option while building `gcc`, the compiler gets built.
- If we do not want to change the configuration system, `spim0.0.md` can be defined to contain

```
(define_insn "dummy_pattern"
    [(reg:SI 0)]
    "1"
    "This stmnt should not be emitted!"
)
```





Operations in Level 0.0

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 |
| RETURN | not required | partial | partial |

spim0.0.c

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

spim0.2.c

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



July 09 Spim MD Levels 0 & 1: Level 0 of Spim Machine Descriptions 20/32

Operations in Level 0

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay

*Part 3*

# *Level 1 of Spim Machine Descriptions*

## 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!



July 09 Spim MD Levels 0 & 1: Level 1 of Spim Machine Descriptions 21/32

Increments for Level 1

Notes

Essential Abstrations 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` $\leftarrow Src$<br>`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` | |



July 09 Spim MD Levels 0 & 1: Level 1 of Spim Machine Descriptions 22/32

Operations Required in Level 1

Notes

Essential Abstrations 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]);

)
```



July 09 Spim MD Levels 0 & 1: Level 1 of Spim Machine Descriptions 23/32

Move Operations in spim1.md

Notes

Essential Abstrations 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")
         (mem:SI (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"
```



July 09 Spim MD Levels 0 & 1: Level 1 of Spim Machine Descriptions 24/32

Move Operations in spim1.md

Notes

Essential Abstrations 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 (mem:SI (match_operand:SI 0 "memory_operand" "m"))
         (match_operand:SI 1 "register_operand" "r"))]
   ""
   "sw \t%1, %m0"
)
```



July 09 Spim MD Levels 0 & 1: Level 1 of Spim Machine Descriptions 25/32

Move Operations in spim1.md

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay



## 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?



July 09 Spim MD Levels 0 & 1: Level 1 of Spim Machine Descriptions 26/32

Using register $zero for constant 0

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay

## 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(...))



July 09 Spim MD Levels 0 & 1: Level 1 of Spim Machine Descriptions 27/32

## Using register $zero for constant 0

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay



## Our Conventions in .md File

- A pattern which is not used for generating the first RTL is preceded by * in its name (eg. "*store_word")
- A pattern with a name preceded by IITB_ is used for generating
  - The specified RTL template by explicitly calling the associated gen function.
  - For example, the RTL template associated with "IITB_move_zero" is generated by calling gen_IITB_move_zero in define_expand "movsi".



July 09 Spim MD Levels 0 & 1: Level 1 of Spim Machine Descriptions 28/32

## Our Conventions in .md File

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay

## 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



July 09 Spim MD Levels 0 & 1: Level 1 of Spim Machine Descriptions 29/32

Supporting Addition in Level 1

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay



## Comparing `movsi` and `addsi3`

- `movsi` uses `define_expand` whereas `addsi3` uses combination of operands
- Why not use constraints for `movsi` too?
- `movsi` combines loads and stores
  - Thus we will need to support memory as both source and destination
  - Will also allow memory to memory move
    Not supported by the machine!



July 09 Spim MD Levels 0 & 1: Level 1 of Spim Machine Descriptions 30/32

Comparing `movsi` and `addsi3`

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay

## Choices for Mapping Compound Operations to Primitive Operations

| Gimple Operation | First RTL | RTL for Scheduling | Assembly | MD Construct |
|---|---|---|---|---|
| Compound | Split | Split | Split | `define_expand` |
| Compound | Compound | Split | Split | `define_split` |
| Compound | Compound | Compound | Split | ASM string |

- `define_expand` may be used for selecting one alternative from among multiple possibilities at the time of first RTL generation.
- `@` in ASM string achieves the same effect at the time of emitting assembly instruction.



July 09 Spim MD Levels 0 & 1: Level 1 of Spim Machine Descriptions 31/32

Choices for Mapping Compound Operations to Primitive Operations

Notes

Essential Abstrations in GCC GCC Resource Center, IIT Bombay

*Part 4*

# *Lab Exercises*



## Lab exercises on Spim Levels 0 & 1

Will be given in the lab :-)

- You may need to refer to chapter 16 and 17 in the GCC Internals document