MITSUBISHI ELECTRIC
Changes for the Better

Automated Generation of Dynamic Binary Translators for Instruction Set Simulation

Katsumi Okuda

MITSUBISHI ELECTRIC CORPORATION

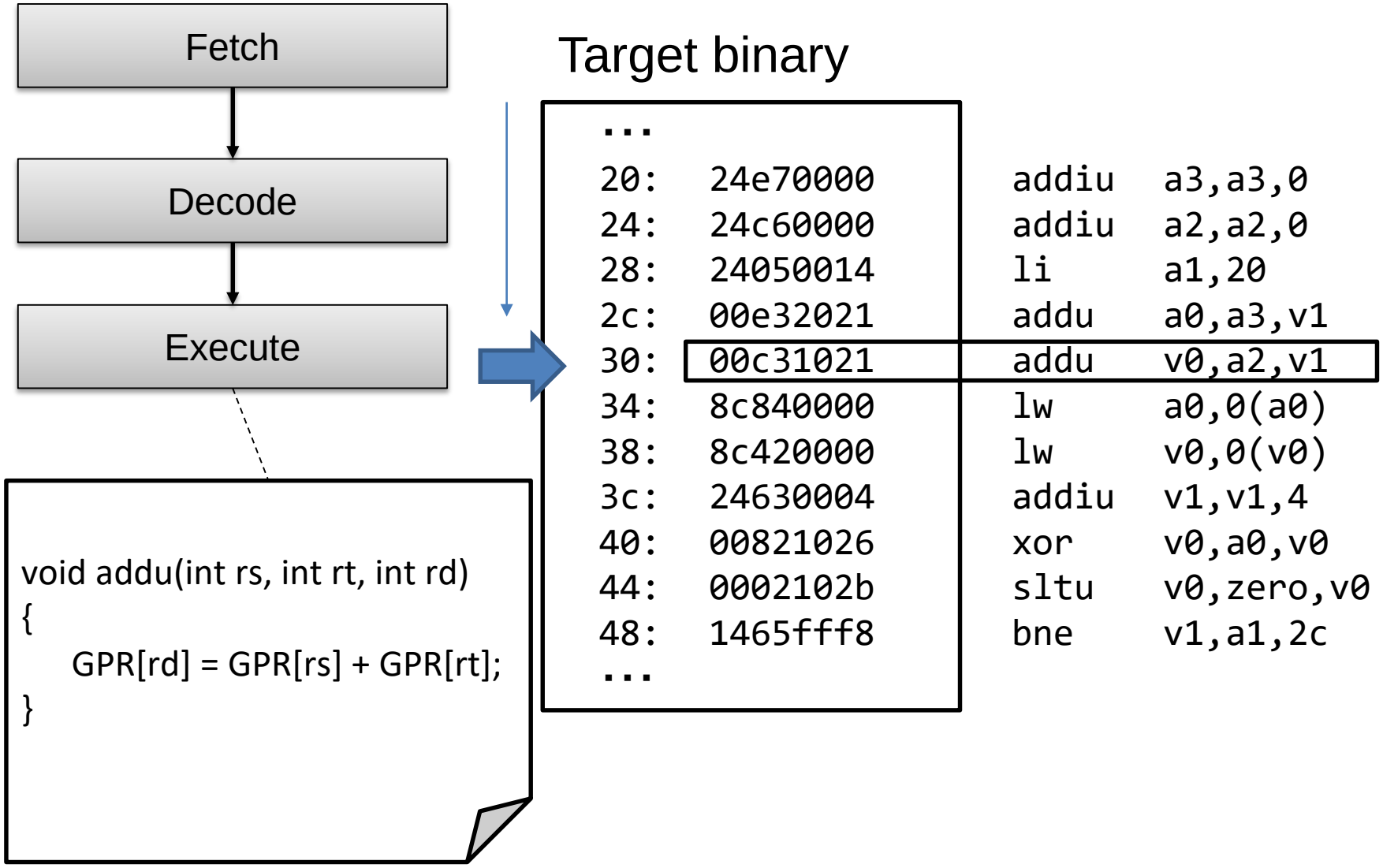
Outline

- Introduction
- Interpreter
- Dynamic binary translator
- Related work
- Generation method
- Experimental results
- Conclusion

Introduction

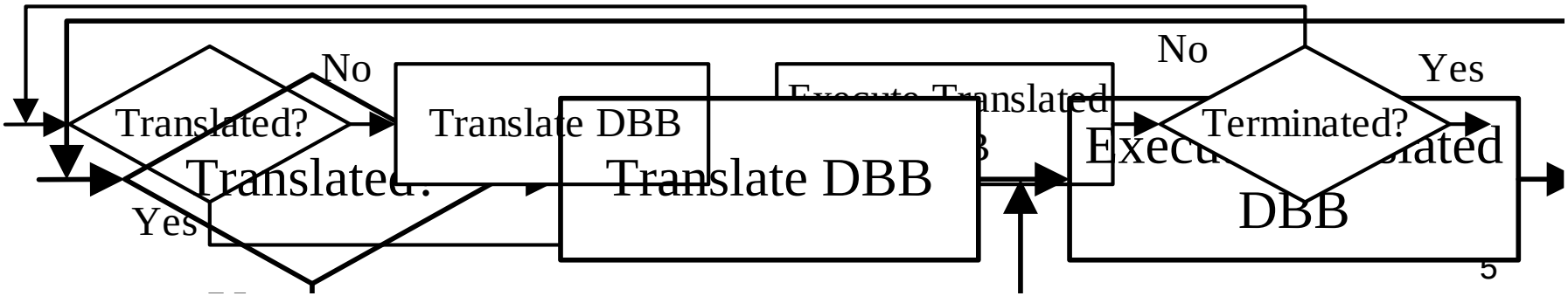
- Background
  - Instruction set simulators are indispensable tools for developing new architecture and embedded software
  - Simulation speed is important
  - ISS must be available at an early stage of development
  - There are two types of widely used ISSs:
    - DBTs(Dynamic binary translators) that are fast and complex
    - Interpreters that are slow and simple
- Goal
  - To generate fast DBTs from descriptions of interpreters

Interpreter



DBT (1/2)

- DBTs translate target instructions into host instructions and execute them
- Resulting host instructions are cached in the translation cache
- The unit of translation is a DBB (Dynamic Basic Block)
- A DBB is an area that begins the instruction that was executed immediately after a branch and ends with the next branch



DBT (2/2)

Target instructions

```
...
20:   24e70000        addiu   a3,a3,0
24:   24c60000        addiu   a2,a2,0
28:   24050014        li      a1,20
2c:   00e32021        addu    a0,a3,v1
30:   00c31021        addu    v0,a2,v1
34:   8c840000        lw      a0,0(a0)
38:   8c420000        lw      v0,0(v0)
3c:   24630004        addiu   v1,v1,4
40:   00821026        xor     v0,a0,v0
44:   0002102b        sltu    v0,zero,v0
48:   1465fff8        bne     v1,a1,2c
...
```

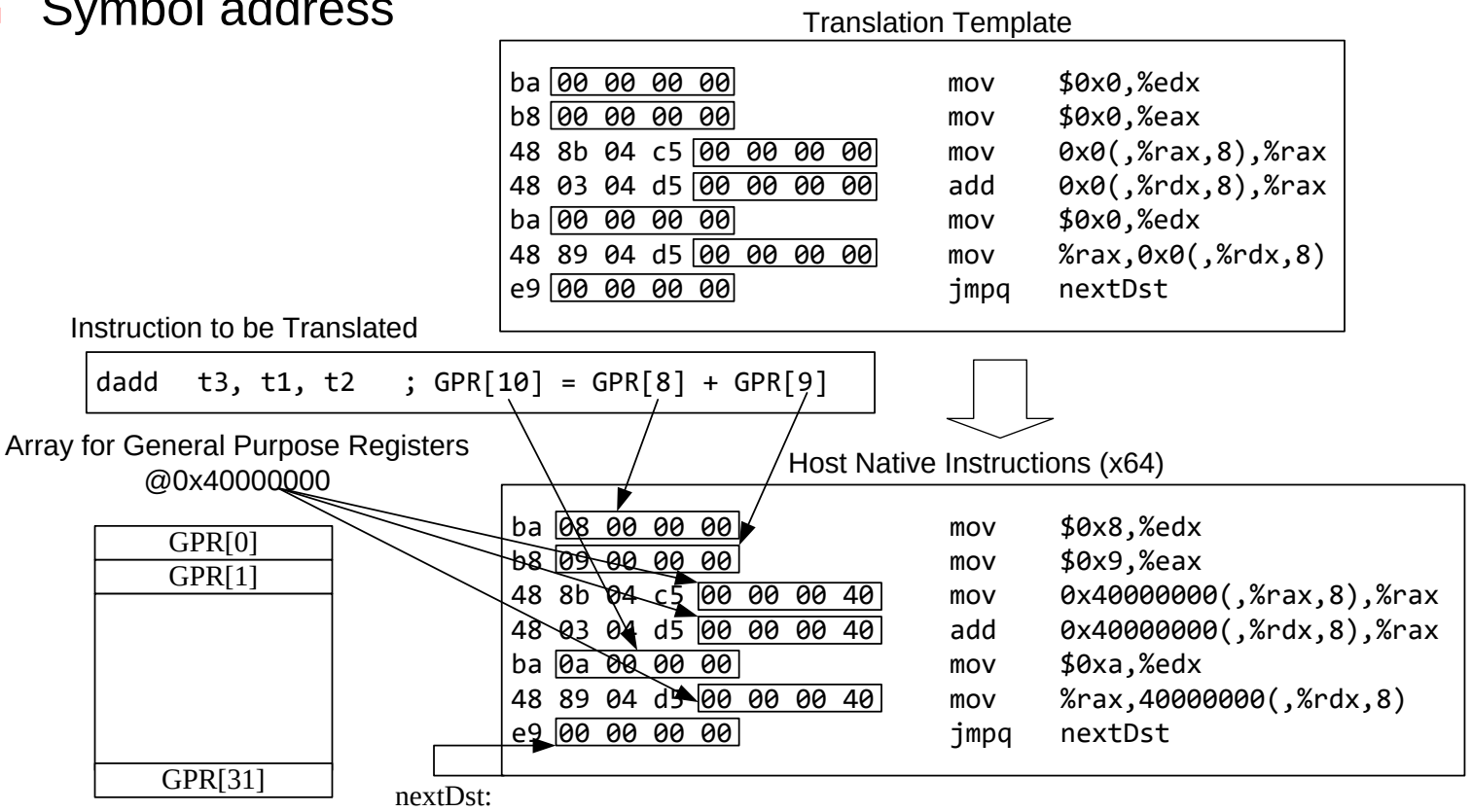
Translate

Host instructions

```
4004be:   8b 15 f8 04 20 00   mov   0x2004f8(%rip),%edx
4004c4:   8b 05 de 04 20 00   mov   0x2004de(%rip),%eax
4004ca:   01 d0               add   %edx,%eax
4004cc:   89 05 de 04 20 00   mov   %eax,0x2004de(%rip)
4004d2:   8b 15 e0 04 20 00   mov   0x2004e0(%rip),%edx
4004d8:   8b 05 d2 04 20 00   mov   0x2004d2(%rip),%eax
4004de:   01 d0               add   %edx,%eax
4004e0:   89 05 c2 04 20 00   mov   %eax,0x2004c2(%rip)
4004e6:   8b 05 c4 04 20 00   mov   0x2004c4(%rip),%eax
4004ec:   89 c7               mov   %eax,%edi
4004ee:   e8 b9 ff ff ff      callq 4004ac <load>
4004f3:   89 05 b7 04 20 00   mov   %eax,0x2004b7(%rip)
4004f9:   8b 05 a9 04 20 00   mov   0x2004a9(%rip),%eax
4004ff:   89 c7               mov   %eax,%edi
400501:   e8 a6 ff ff ff      callq 4004ac <load>
          89 05 9c 04 20 00   mov   %eax,0x20049c(%rip)
          05 9a 04 20 00      mov   0x20049a(%rip),%eax
          c0 04               add   $0x4,%eax
          89 05 91 04 20 00   mov   %eax,0x200491(%rip)
40051b:   8b 15 8f 04 20 00   mov   0x20048f(%rip),%edx
400521:   8b 05 81 04 20 00   mov   0x200481(%rip),%eax
400527:   31 d0               xor   %edx,%eax
400529:   89 05 79 04 20 00   mov   %eax,0x200479(%rip)
40052f:   8b 05 73 04 20 00   mov   0x200473(%rip),%eax
400535:   c1 e8 1f            shr   $0x1f,%eax
400538:   0f b6 c0            movzbl %al,%eax
40053b:   89 05 67 04 20 00   mov   %eax,0x200467(%rip)
400541:   8b 15 65 04 20 00   mov   0x200465(%rip),%edx
400547:   8b 05 67 04 20 00   mov   0x200467(%rip),%eax
40054d:   39 c2               cmp   %eax,%edx
40054f:   74 0a               je    40055b <doit+0xa1>
400551:   c7 05 c5 04 20 00 2c movl  $0x2c,0x2004c5(%rip
```

Translation template

- DBTs translate target instructions one by one
- DBTs use a translation template to translate an instruction
- Translation templates have the following parameters:
  - Instruction field
  - Program counter
  - Symbol address

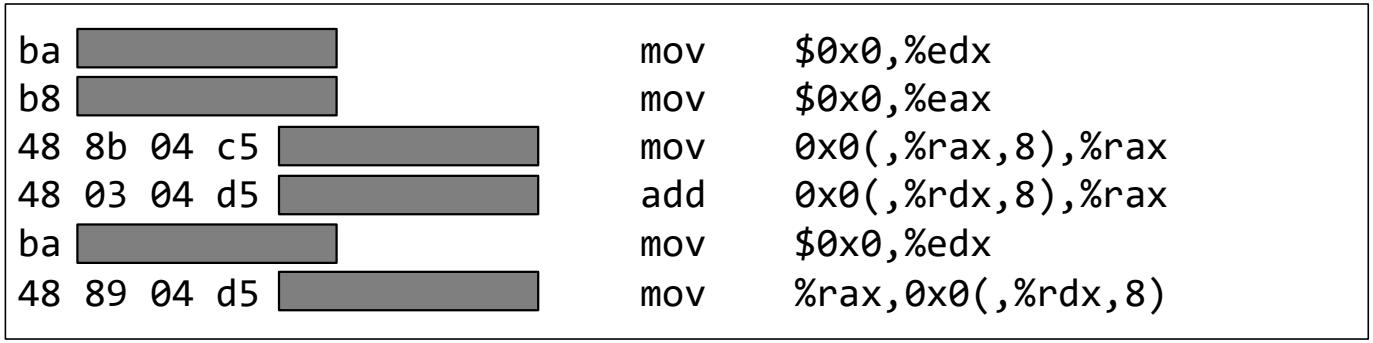


Basic Idea

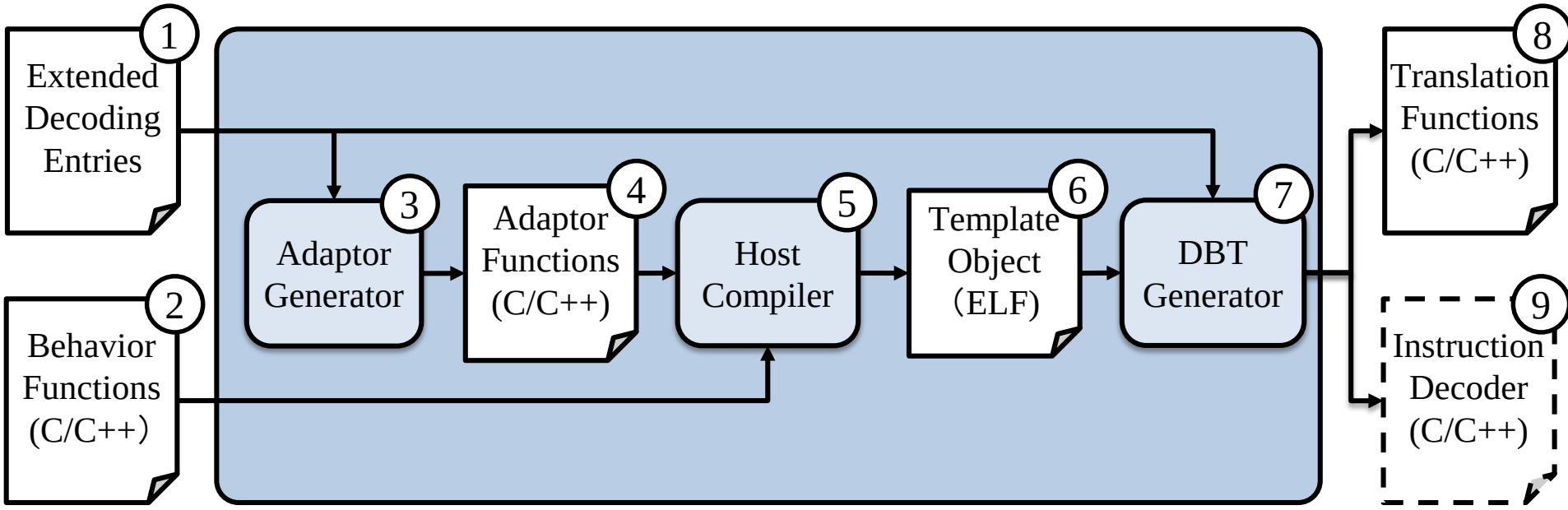
- The creation of translation templates needs a large effort, since the developer need to manipulate the host instructions.
- We solve this problem by generating translation template from behavior functions written in C/C++.
- A behavior function is a behavior description for an interpreter and not a translation description.

```
DEFINST(DADD)
{
    GPR[rd] = GPR[rs] + GPR[rt];
}
```

Compile and extract a template



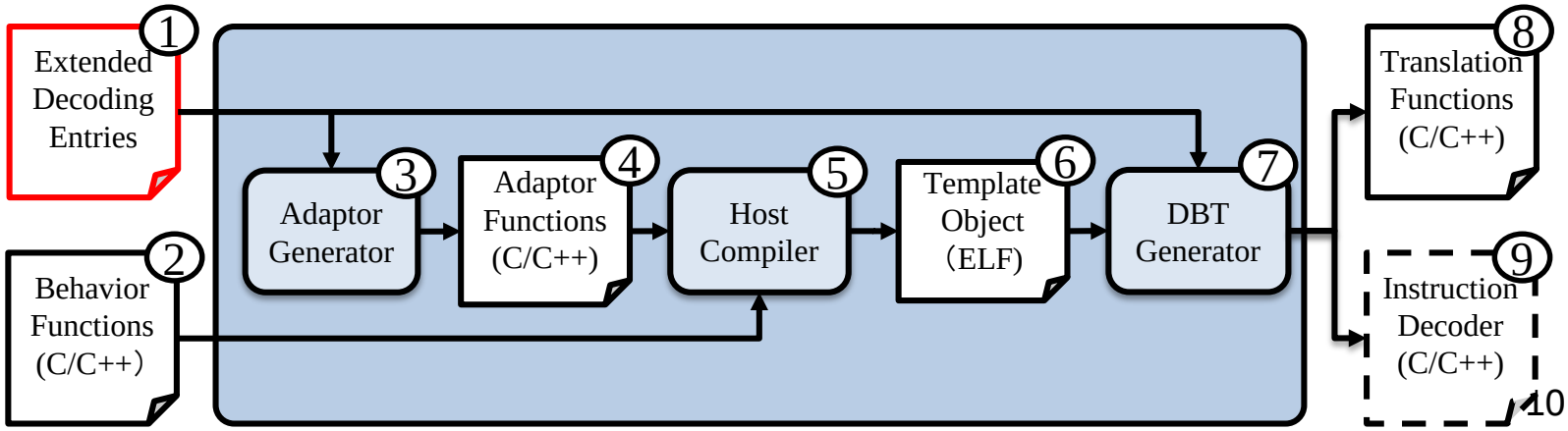
Generation flow



Extended decoding entry

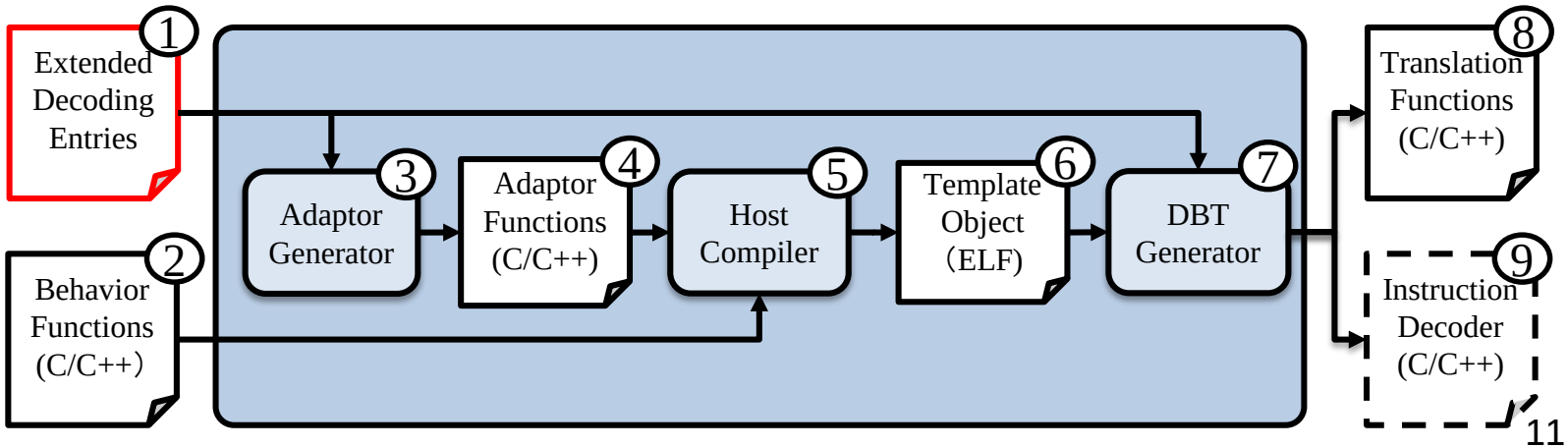
- An extended decoding entry is an extended version of a decoding entry for generation of instruction decoders.
- Each instruction has a decoding entry for it.
- An entry have the following fields:
  - Instruction name
  - pattern
  - Exclusion conditions if any
  - Instruction fields
  - Instruction type
  - Branch type
  - Delay slot count

} for generation of instruction decoders



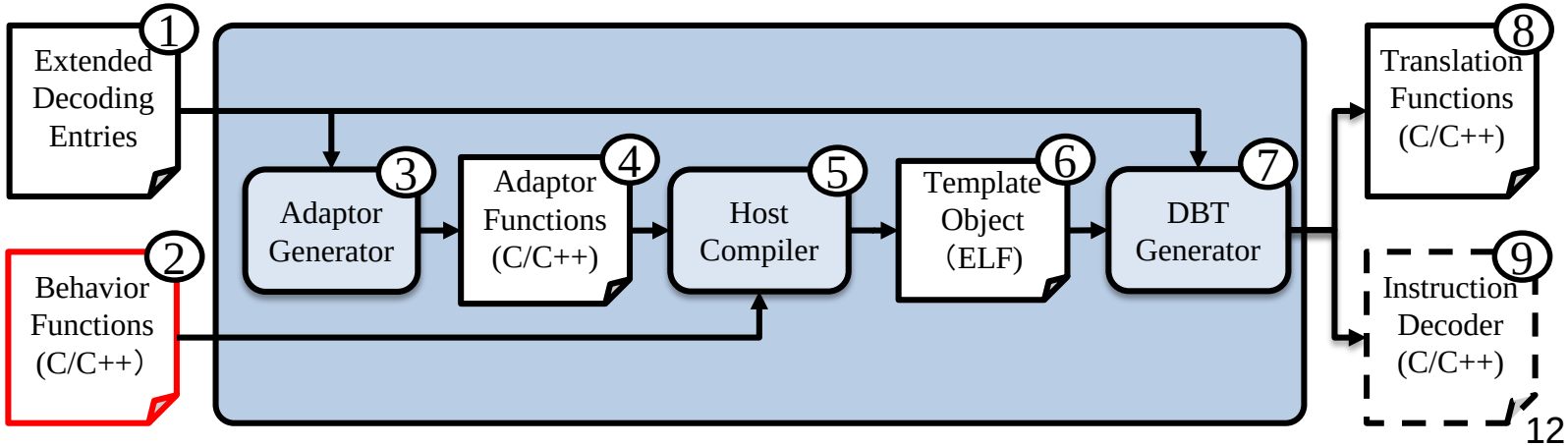
Examples of decoding entries

Field	BL (ARM)	DADD (MIPS64)
Instruction name	BL	DADD
Pattern	xxxx1011xxxxxxxxxxxxxxxxxxxxxxxx	000000xxxxxxxxxxxxxxx00000101100
Exclusion conditions	cond = 1111	-
Fields	cond[31:28], imm24[23:0]	rs[25:21], rt[20:16], rd[15:11]
Branch	True	False
Conditional	True	-
Delay slot count	0	-



Behavior function

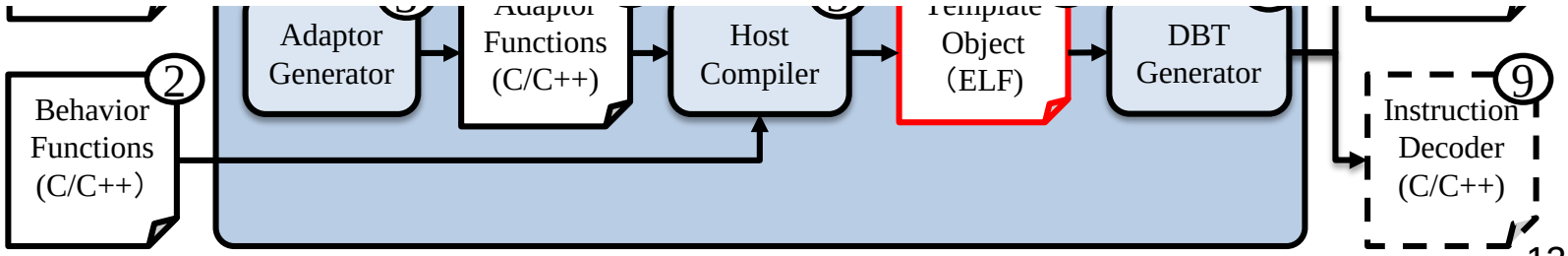
```
1   void DADD(uint32_t pc, uint32_t rs, uint32_t rt,
        uint32_t rd)
2   {
3     GPR[rd] = GPR[rs] + GPR[rt];
4   }
5
6   void BEQ(uint32_t pc, uint32_t rs, uint32_t rt,
        uint32_t offset)
7   {
8     BRANCH_RESULT = GPR[rs] == GPR[rt];
9     if (BRANCH_RESULT) {
10      NEXT_PC = pc + (sext16(offset) << 2);
11    }
12  }
```



Adaptor function

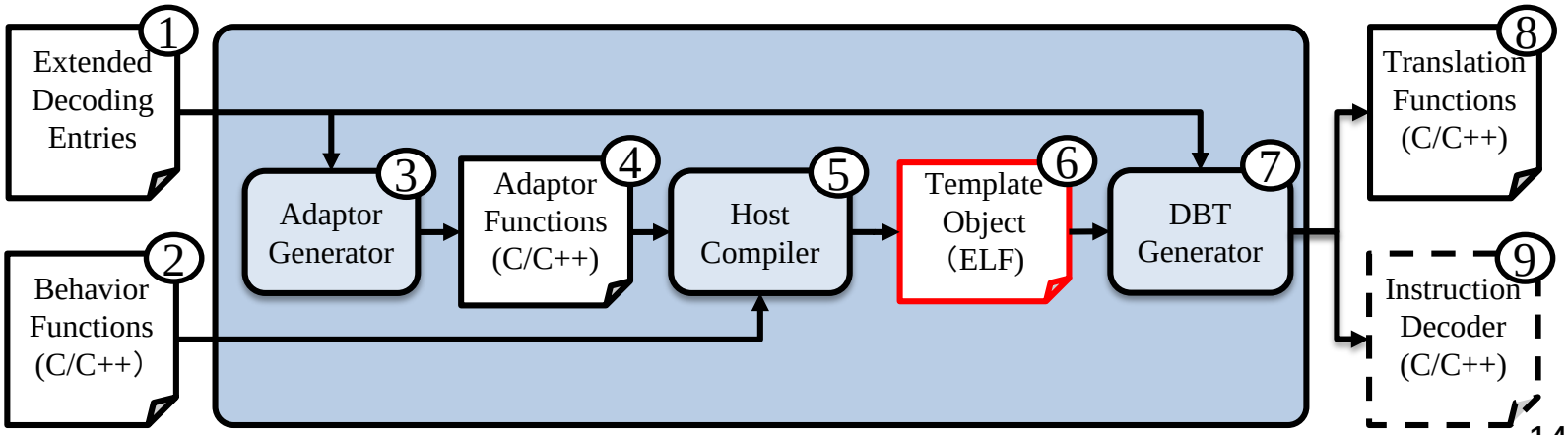
- An adaptor function is the wrapper of a behavior function.
- Adaptor functions enables the parameters of behavior function to be relocatable symbols in the resulting object file.
- An adaptor function have markers that helps a DBT generator extract the translation template from an template object.

```
1  void AdaptorDADD(void)
2  {
3      extern int pc, i_rs, i_rt, i_rd;
4
5      MARK;
6      DADD((uint32_t)&pc, (uint32_t)&i_rs, (uint32_t)&
           i_rt, (uint32_t)&i_rd);
7      JUMP_TO_NEXT_BLOCK;
8  }
```



Template object

- A template objects is a compiling result of adaptor functions
- The format of a template object is a relocatable object file format such as ELF which has a relocation table
- A relocation table provides the information to get the positions of parameters in translation templates



Translation function

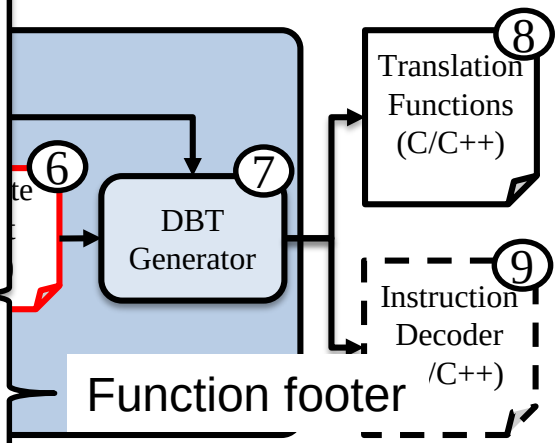
```
1  /* Start: Function Header */
2  void TranslateDADD(uint32_t pc, uint32_t rs, uint32_t
       rt, uint32_t rd)
3  {
4  /* End: Function Header */
5
6    /* Start: Template Expansion */
7    static const uint8_t template[] = {
8        0xBA, 0x00, 0x00, 0x00, 0x00, ...
9        ...
10       ..., 0x00, 0x00, 0x00, 0xC3 };
11   memcpy(dst, template, sizeof(template));
12   uint64_t nextDst = (uint64_t)dst + sizeof(template);
13   /* End: Template Expansion */
14
15   /* Start: Parameter Substitutions
16   memcpy(&dst[1], &rs, sizeof(rs));
17   memcpy(&dst[6], &rt, sizeof(rt));
18   uint32_t tmp1 = (uint32_t)&GPR;
19   memcpy(&dst[14], &tmp1, sizeof(tmp1));
20   ...
21   uint32_t tmp2 = uint32_t(nextDst + -4 - (uint64_t)&
         dst[51]);
22   memcpy(&dst[51], &tmp2, sizeof(tmp2));
23   /* End: Parameter Substitutions */
24
25 /* Start: Function Footer */
26   dst = nextDst;
27 }
28 /* End: Function Footer */
```

Function header

Template expansion

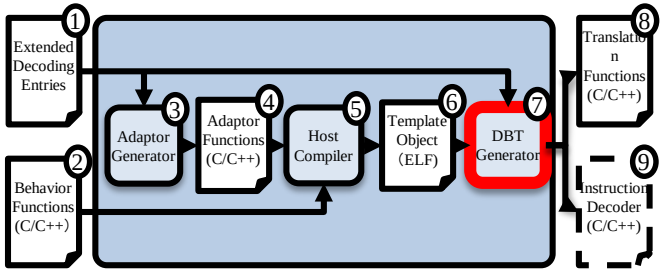
Parameter substations

Function footer



Generation algorithm

```
1  /* Start: Function Header */
2  void TranslateDADD(uint32_t pc, uint32_t rs, uint32_t
       rt, uint32_t rd)
3  {
4  /* End: Function Header */
5
6    /* Start: Template Expansion */
7    static const uint8_t template[] = {
8        0xBA, 0x00, 0x00, 0x00, 0x00, ...
9        ...
10       ..., 0x00, 0x00, 0x00, 0xC3 };
11   memcpy(dst, template, sizeof(template));
12   uint64_t nextDst = (uint64_t)dst + sizeof(template);
13   /* End: Template Expansion */
14
15   /* Start
16   memcpy(&
17   memcpy(&
18   uint32_t
19   memcpy(&
20   ...
21   uint32_t
         dst
22   memcpy(&
23   /* End:
24
25 /* Start:
26   dst = ne
27 }
28 /* End: F
```



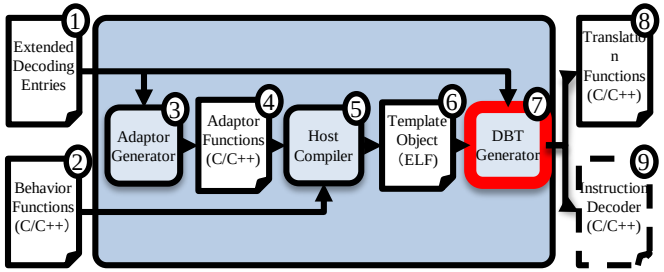
Algorithm 1 Generation algorithm

Input: Template object *objFile*, Decoding entry *entry*
Output: Translation Function

1: *CreateFunctionHeader*(*entry.name*, *entry.fields*)
2: *symbol* ← *FindSymbol*(*objFile.symbolTable*, *entry.name*)
3: *CreateTemplateExpansion*(*symbol*)
4: **for all** *relocationEntry* in *objFile.relocationTable* **do**
5: **if** *IsParameter*(*symbol*, *relocationEntry*) **then**
6: *CreateParameterSubstituion*(*symbol*, *relocationEntry*)
7: *CreateFunctionFooter*()

Generation algorithm

```
1  /* Start: Function Header */
2  void TranslateDADD(uint32_t pc, uint32_t rs, uint32_t
       rt, uint32_t rd)
3  {
4  /* End: Function Header */
5
6    /* Start: Template Expansion */
7    static const uint8_t template[] = {
8        0xBA, 0x00, 0x00, 0x00, 0x00, ...
9        ...
10       ..., 0x00, 0x00, 0x00, 0xC3 };
11   memcpy(dst, template, sizeof(template));
12   uint64_t nextDst = (uint64_t)dst + sizeof(template);
13   /* End: Template Expansion */
14
15   /* Start
16   memcpy(&
17   memcpy(&
18   uint32_t
19   memcpy(&
20   ...
21   uint
         ds
22   memcpy(&
23   /* End:
24
25 /* Start:
26   dst = ne
27 }
28 /* End: F
```



Algorithm 1 Generation algorithm

Input: Template object *objFile*, Decoding entry *entry*
Output: Translation Function

1: $CreateFunctionHeader(entry.name, entry.fields)$
2: $symbol \leftarrow FindSymbol(objFile.symbolTable, entry.name)$
3: $CreateTemplateExpansion(symbol)$
4: **for all** $relocationEntry$ in $objFile.relocationTable$ **do**
5: **if** $IsParameter(symbol, relocationEntry)$ **then**
6: $CreateParameterSubstituion(symbol, relocationEntry)$
7: $CreateFunctionFooter()$

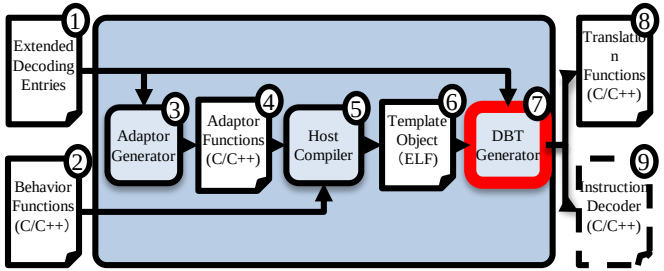
Generation algorithm

```
1  /* Start: F
2  void Transla
       rt, uin
3  {
4  /* End: Fun
5
6    /* Start:
7    static co
8        0xBA
9          ...
10         ...,
11   memcpy(ds
12   uint64_t
13   /* End: T
14
15   /* Start:
16   memcpy(&dst[1], &rs, sizeof(rs));
17   memcpy(&dst[6], &rt, sizeof(rt));
18   uint32_t tmp1 = (uint32_t)&GPR;
19   memcpy(&dst[14], &tmp1, sizeof(tmp1));
20   ...
21   uint32_t tmp2 = uint32_t(nextDst + -4 - (uint64_t)&
         dst[51]);
22   memcpy(&dst[51], &tmp2, sizeof(tmp2));
23   /* End: Parameter Substitutions */
24
25 /* Start: Function Footer */
26   dst = nextDst;
27 }
28 /* End: Function Footer */
```

Algorithm 1 Generation algorithm

Input: Template object *objFile*, Decoding entry *entry*
Output: Translation Function

1: $CreateFunctionHeader(entry.name, entry.fields)$
2: $symbol \leftarrow FindSymbol(objFile.symbolTable, entry.name)$
3: $CreateTemplateExpansion(symbol)$
4: **for all** $relocationEntry$ in $objFile.relocationTable$ **do**
5: **if** $IsParameter(symbol, relocationEntry)$ **then**
6: $CreateParameterSubstituion(symbol, relocationEntry)$
7: $CreateFunctionFooter()$



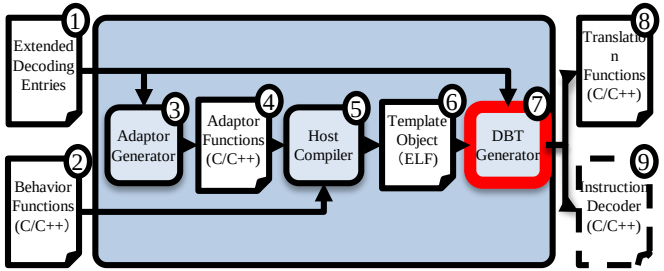
Generation algorithm

Algorithm 1 Generation algorithm

Input: Template object *objFile*, Decoding entry *entry*
Output: Translation Function

1: *CreateFunctionHeader*(*entry.name*, *entry.fields*)
2: *symbol* ← *FindSymbol*(*objFile.symbolTable*, *entry.name*)
3: *CreateTemplateExpansion*(*symbol*)
4: **for all** *relocationEntry* in *objFile.relocationTable* **do**
5: **if** *IsParameter*(*symbol*, *relocationEntry*) **then**
6: *CreateParameterSubstituion*(*symbol*, *relocationEntry*)
7: *CreateFunctionFooter*()

```
1  /* Start: F
2  void Transla
       rt, uin
3  {
4  /* End: Fun
5
6    /* Start:
7    static co
8        0xBA
9          ...
10         ...,
11   memcpy(ds
12   uint64_t
13   /* End: T
14
15   /* Start:
16   memcpy(&dst[1], &rs, sizeof(rs));
17   memcpy(&dst[6], &rt, sizeof(rt));
18   uint32_t tmp1 = (uint32_t)&GPR;
19   memcpy(&dst[14], &tmp1, sizeof(tmp1));
20   ...
21   uint32_t tmp2 = uint32_t(nextDst + -4 - (uint64_t)&
         dst[51]);
22   memcpy(&dst[51], &tmp2, sizeof(tmp2));
23   /* End: Parameter Substitutions */
24
25 /* Start: Function Footer */
26   dst = nextDst;
27 }
28 /* End: Function Footer */
```



Experimental setup

- We implemented DBTs for ARM, SH, MIPS64 using the proposed framework and measured the productivity and the performance.
- Productivity
  - We compared our ISS and GDB's ISS in terms of total amount of the description for ISSs.
- Performance
  - We measured the performance of generated DBTs using
  - We also measured the speedup of DBTs from their original interpreters.

Productivity

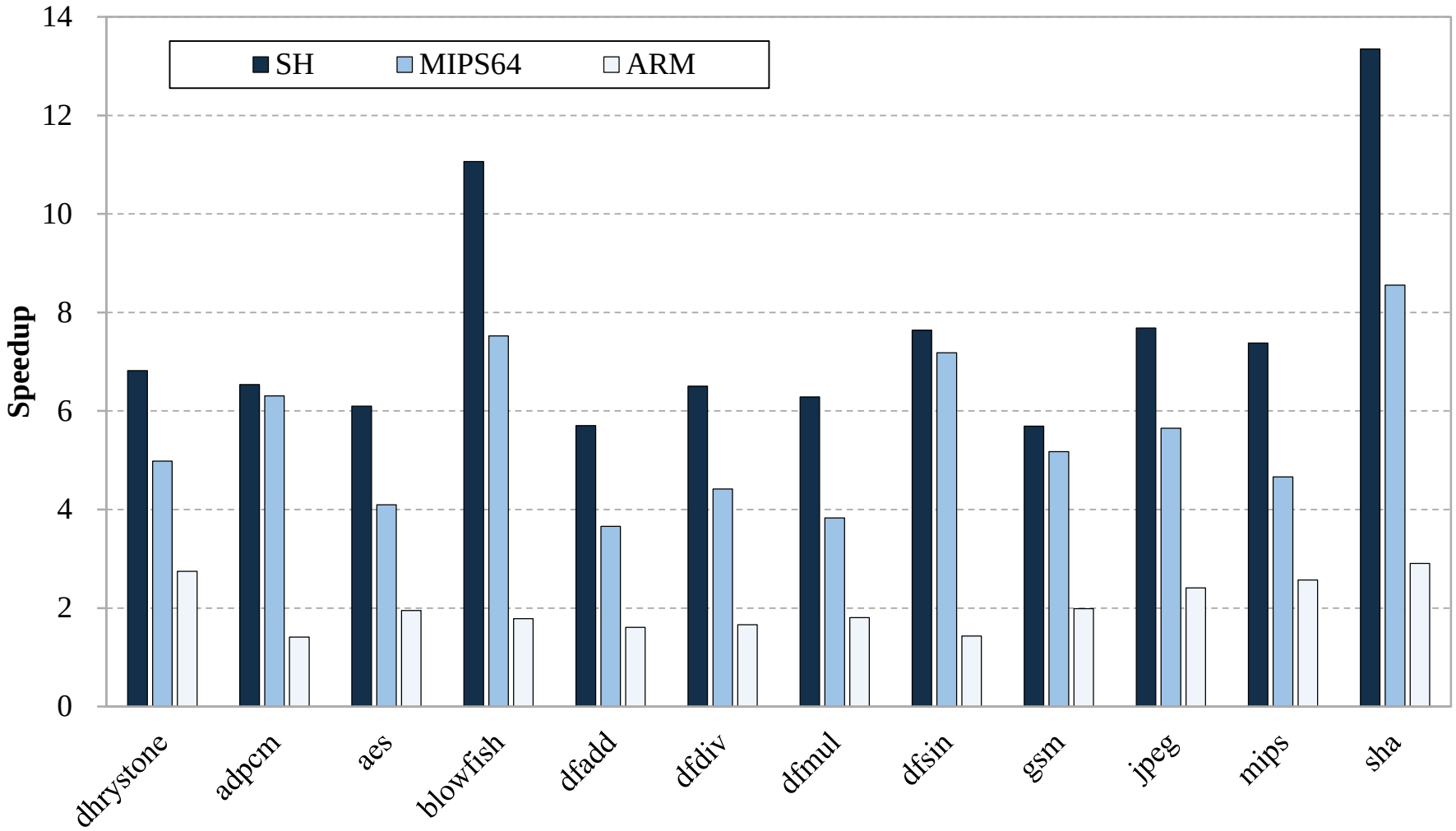
	Our framework			GDB
	SH	MIPS64	ARM	ARM
Implemented instructions	152	234	175	170
LOC	870	1,257	2,111	3,461
LOC per instruction	6	5	12	20

Performance (CPI)

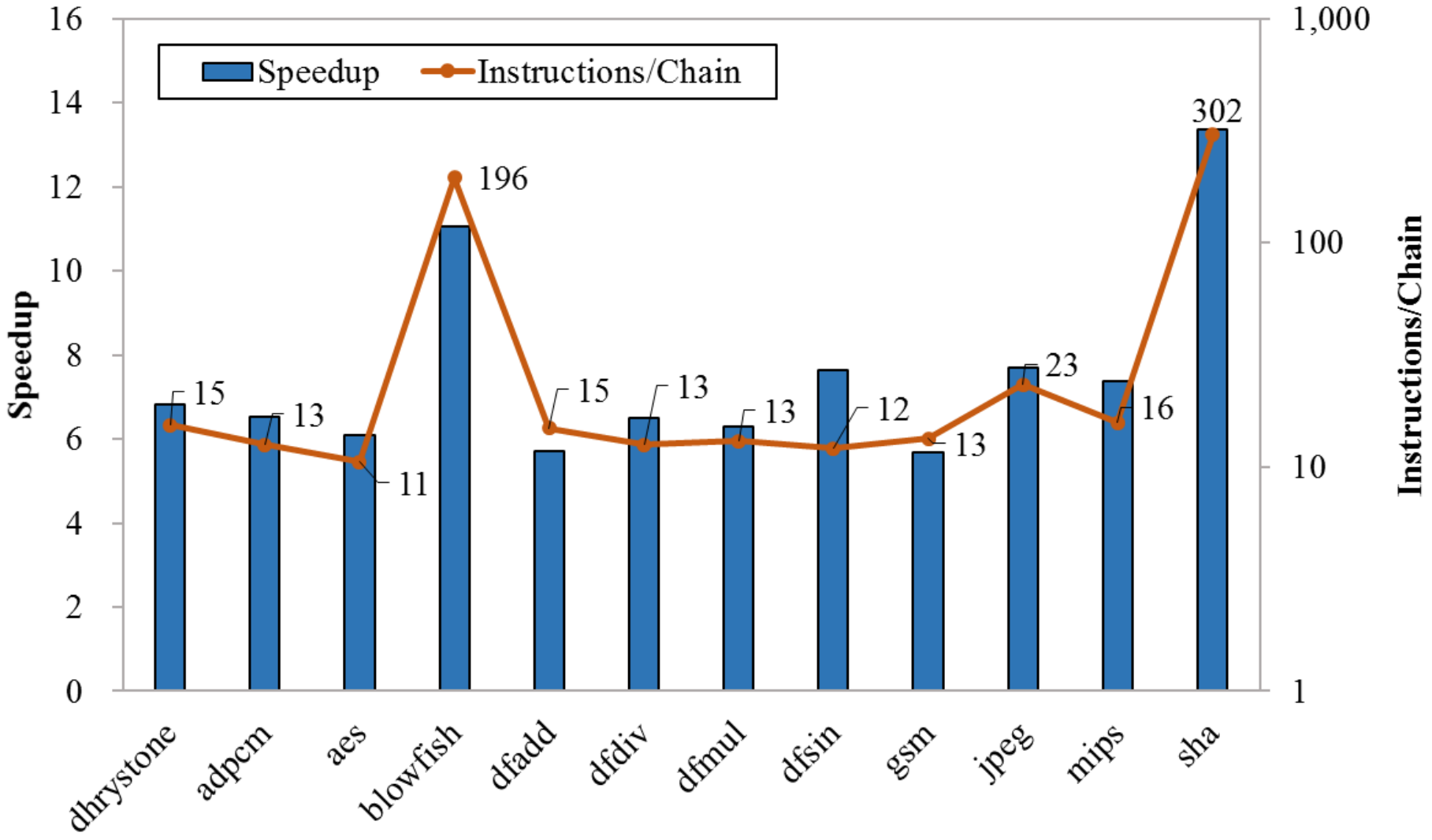
- CPI: host Cycles per target Instructions

Benchmark	Interpreter			DBT		
	SH	MIPS64	ARM	SH	MIPS64	ARM
dhrystone	61.95	49.86	76.16	9.09	10.00	27.74
adpcm	53.95	52.76	69.95	8.25	8.37	49.56
aes	52.54	46.24	72.80	8.62	11.29	37.30
blowfish	52.46	43.23	72.45	4.74	5.75	40.60
dfadd	69.66	47.93	88.87	12.22	13.10	55.22
dfdiv	66.41	53.54	79.45	10.21	12.12	47.78
dfmul	62.78	53.64	88.57	9.99	14.01	49.12
dfsin	67.09	51.41	76.67	8.78	7.16	53.51
gsm	61.95	49.86	70.23	8.93	7.41	35.38
jpeg	54.35	47.82	71.77	7.07	8.46	29.77
mips	47.21	40.14	63.94	6.40	8.61	24.89
motion	56.44	59.31	73.42	16.93	25.75	41.02
sha	54.86	41.69	71.55	4.11	4.87	24.63

Speedup


SH
MIPS64
ARM
Speedup
14
12
10
8
6
4
2
0
dhrystone
adpcm
aes
blowfish
dfadd
dfdiv
dfmul
dfsin
gsm
jpeg
mips
sha

Relation between speedup and DBB length



Conclusion

- The proposed generation framework:
  - addresses the difficulty of implementing a DBT
  - can generate DBTs that is 1.4 to 13.4 timers faster than their original interpreter
  - can be used with an existing instruction decoder generation algorithm