EBC Compiler

Ravi Narayanaswamy
Jiang Ning Liu

Disclaimer

THIS INFORMATION COTNAINED IN THIS DOCUMENT, INCLUDING ANY TEST RESULTS ARE PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT OR BY THE SALE OF INTEL PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel retains the right to make changes to its specifications at any time, without notice.

Recipients of this information remain solely responsible for the design, sale and functionality of their products, including any liability arising from product infringement or product warranty.

Intel may make changes to specifications, product roadmaps and product descriptions at any time, without notice.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Agenda

- Source Language
- Virtual Machine
- Development Environment
- EBC Compiler
- Demos

Source Language

- Run on various platform
  - 64-bit Clean
- Small image
  - No floating point types/operations
  - No C++ (runtime library, exception handling)
- Debuggability/Maintenance
  - No inline asm

Source Language -64bitness

- Natural Types
  - Pointer
  - INTN/UINTN
- No static initialization using sizeof on natural types

Language Examples(1)

```
main()
{
  int *xp, x, y;

  y = 8;
  xp = &y;
  x = *xp;
}
```

Language Examples(2)

```
int PASS;
long f(long x)
{
  return x+sizeof(int);
}
main()
{
  long x, y;

  x=4;
  y = f(x);
  if ((y+f(x))==16)
  PASS=1;
  else PASS=0;
}
```

Virtual Machine

- Registers
- Instruction encoding
- Operand Register encoding
- Natural indexing
- Instruction Set

Registers

- General purpose Registers
  - 8 - 64 bit registers (R0 – R7)
- Special Registers
  - 64 bit registers
  - Currently only 2 defined

Registers cont.

- General purpose Registers

Index	Register	Description
0	R0	Points to the top of the stack
1-3	R1-R3	Preserved across calls
4-7	R4-R7	Scratch, not preserved across calls

- Special Registers

Index	Register	Description
0	FLAGS	0 : Condition Code, 1..63 Reserved
1	IP	Points to current instruction
2..7	Reserved	Not defined

Instruction Encoding

- Opcode operand1 operand2

Bit	Sym	Description
0-5	Op	The opcode of the instruction
6	W	Width
7	I	Immediate data Present

Operand Register Encoding

- [@]Rn

Bit	Description
0..2	Operand 1 Register
3	0 = Operand 1 is direct 1 = Operand 1 is indirect
4..6	Operand 2 register
7	0 = Operand 2 is direct 1 = Operand 2 is indirect

Natural Indexing

- Immediate data for indirect operands
- Index = C + N * (Size of pointer in bytes)
- Indexes can be 16, 32 or 64 bit wide

Bit	Description
x+4	Sign bit, most significant bit
x+1..x+3	Bits assigned to natural units (w)
a+1..x	Constant units (C)
0..a	Natural units (N)

Instruction Set

- Program Flow
- Compare
- Data Manipulation
- Data Movement

Program Flow Instructions

- BREAK
- JMP
- CALL
- RETURN

Program Flow Instructions

- Break break_code

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>5..7</td><td>Reserved, must be 0</td></tr>
<tr><td>0..4</td><td>Opcode=0</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>Break Code</td></tr>
</table>

Program Flow Instructions

- Break Code

Code	Description
0	Runaway program break
1	VM Revision number
2	Skip
3	Debug Break.
4	System Call.
5	Create Thunk
6	Set Compiler Version
7..255	Reserved

Program Flow Instructions

- JMP64 [cs|cc] Imm64
- JMP32 [cs|cc], [@]R1 [Imm32]
- JMP8 [cs|cc] Imm8

Program Flow Instructions

- JMP

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>1=Immediate data present</td></tr>
<tr><td>6</td><td>0=32 bit immediate data 1 = 64 bit immediate data</td></tr>
<tr><td>0..5</td><td>Opcode 1</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>0= unconditional 1= conditional</td></tr>
<tr><td>6</td><td>0= CC 1=CS</td></tr>
<tr><td>5</td><td>Reserved</td></tr>
<tr><td>4</td><td>0= Absolute address 1= Relative address</td></tr>
<tr><td>3</td><td>0 = Operand1 direct 1= Operand1 indirect</td></tr>
<tr><td>0..2</td><td>Operand1</td></tr>
</table>
</td></tr>
<tr><td>2..5</td><td>Optional 32 bit immediate data</td></tr>
<tr><td>2..7</td><td>Optional 64 bit immediate data</td></tr>
</table>

Program Flow Instructions

- JMP8

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>0= Unconditional
1=Conditional</td></tr>
<tr><td>6</td><td>0=CC 1=CS</td></tr>
<tr><td>0..5</td><td>Opcode 2</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>8 bit offset</td></tr>
</table>

Program Flow Instructions

- CALL32 [EX] [a] [@] R1 [Imm32]
- CALL64 [EX] [a] Imm64

Program Flow Instructions

- CALL

Byte	Description
0	<table><tr><th>Bit</th><th>Description</th></tr><tr><td>7</td><td>1=Immediate data present</td></tr><tr><td>6</td><td>0=Call32 1 = Call64</td></tr><tr><td>0..5</td><td>Opcode 3</td></tr></table>
1	<table><tr><th>Bit</th><th>Description</th></tr><tr><td>6-7</td><td>Reserved, must be 0</td></tr><tr><td>5</td><td>0= Call to EBC code 1= Call to Native code</td></tr><tr><td>4</td><td>0= Absolute address 1= Relative address</td></tr><tr><td>3</td><td>0 = Operand1 direct 1= Operand1 indirect</td></tr><tr><td>0..2</td><td>Operand1</td></tr></table>
2..3	Optional 32 bit data
2..7	Optional 64 bit immediate data

Program Flow Instructions

- RET

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7..4</td><td>Reserved, must be 0</td></tr>
<tr><td>0..5</td><td>Opcode=4</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>Reserved</td></tr>
</table>

Compare

Opcode	cc	Description
5/45	eq	Compare Signed Equal/Not Equal
6/46	lte	Compare Signed Less Than or Equal/Greater Than
7/47	gte	Compare Signed Greater Than or Equal/Less Than
8/48	ulte	Compare Unsigned Less Than or Equal/Greater Than
9/49	ugte	Compare Unsigned Greater Than or Equal/Less Than

Compare

- CMP[32|64]cc R1,[@]R2[Imm16]

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>0= No immediate data
1=Immediate data present</td></tr>
<tr><td>6</td><td>0=32 bit operand width
1=64 bit operand width</td></tr>
<tr><td>0..5</td><td>CMP opcodes</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>Operand</td></tr>
<tr><td>2..3</td><td>Optional 16 bit immediate data</td></tr>
</table>

Compare

- CMPI[32|64]cc R1,[Imm16|32]

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>0= 16 bit immediate data
1= 32 bit Immediate data</td></tr>
<tr><td>6</td><td>0=32 bit operand width
1=64 bit operand width</td></tr>
<tr><td>0..5</td><td>CMP opcodes</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>Operand</td></tr>
<tr><td>2..3</td><td>16 bit immediate data</td></tr>
<tr><td>2..5</td><td>32 bit immediate data</td></tr>
</table>

Data Manipulation

Opcode	Description	
10	NOT[64|32] R1,[@]R2[Imm16]	R1 = NOT R2
11	NEG[64|32] R1,[@]R2[Imm16]	R1 = NEG R2
12	ADD[64|32] R1,[@]R2[Imm16]	R1 = R1+R2
13	SUB[64|32] R1,[@]R2[Imm16]	R1 = R1-R2
14	MUL[64|32] R1,[@]R2[Imm16]	R1 = R1*R2
15	MULU[64|32] R1,[@]R2[Imm16]	R1 = R1*R2
16	DIV [64|32] R1,[@]R2[Imm16]	R1 = R1/R2
17	DIVU [64|32] R1,[@]R2[Imm16]	R1 = R1/R2
18	MOD [64|32] R1,[@]R2[Imm16]	R1 = R1 mod R2
19	MODU [64|32] R1,[@]R2[Imm16]	R1 = R1 mod R2
20	AND [64|32] R1,[@]R2[Imm16]	R1 = R1 and R2
21	OR [64|32] R1,[@]R2[Imm16]	R1 = R1 or R2

Data Manipulation

Opcode	Description	
22	XOR[64|32] R1,[@]R2[Imm16]	R1 = R1 xor R2
23	SHL[64|32] R1,[@]R2[Imm16]	R1 = R1 shl R2
24	SHR[64|32] R1,[@]R2[Imm16]	R1 = R1 shr R2
25	ASHR[64|32] R1,[@]R2[Imm16]	R1 = R1 ashr R2
26	EXTNDB[64|32] R1,[@]R2[Imm16]	Extract Byte R2, sign extend to 64 bits and store back in R1
27	EXTNDW[64|32] R1,[@]R2[Imm16]	Extract Word R2, sign extend to 64 bits and store back in R1
28	EXTNDD [64|32] R1,[@]R2[Imm16]	Extract Dword R2, sign extend to 64 bits and store back in R1

Data Manipulation

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>0= No immediate data
1= Immediate data present</td></tr>
<tr><td>6</td><td>0=32 bit operand width
1=64 bit operand width</td></tr>
<tr><td>0..5</td><td>Data Manipulation opcodes</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>Operand</td></tr>
<tr><td>2..3</td><td>Optional 16 bit immediate data</td></tr>
</table>

Data Movement

- MOV[s][n][B|W|D|Q][W|D|Q] [@]R1,[@]R2[Imm16|32|64]
- MOV[s][n][B|W|D|Q][W|D|Q] [@]R1[Imm16|32|64],[@]R2
- MOVI[B|W|D|Q][W|D|Q] [@]R1[Imm16],Imm16|32|64
- MOVIn[W|D|Q] [@]R1[Imm16],Imm16|32|64
- MOVREL{W|D|Q] [@]R1[Imm16],Imm16|32|64

Data Movement

Opcode	Description
29	MOVbw [@]R1[Imm16],[@]R2[Imm16]
30	MOVww [@]R1[Imm16],[@]R2[Imm16]
31	MOVdw [@]R1[Imm16],[@]R2[Imm16]
32	MOVqw [@]R1[Imm16,[@]R2[Imm16]
33	MOVbd [@]R1[Imm32],[@]R2[Imm32]
34	MOVwd [@]R1 [Imm32],[@]R2[Imm32]
35	MOVdd [@]R1 [Imm32],[@]R2[Imm32]
36	MOVqd [@]R1 [Imm32],[@]R2[Imm32]

Data Movement

Opcode	Description
37	MOVsnw [@]R1[Imm16],[@]R2[Imm16]
38	MOVsnd [@]R1[Imm32],[@]R2[Imm32]
40	MOVqq [@]R1[Imm64],[@]R2[Imm64]
50	MOVnw [@]R1[Imm16],[@]R2[Imm16]
51	MOVnd [@]R1[Imm32,[@]R2[Imm32]

Data Movement

MOV/MOVn

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>0= Operand 1 index absent
1= Operand 1 index present</td></tr>
<tr><td>6</td><td>0= Operand 2 index absent
1= Operand 2 index present</td></tr>
<tr><td>0..5</td><td>Mov opcodes</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>Operand</td></tr>
<tr><td>2..3</td><td>Optional 16 bit immediate data</td></tr>
<tr><td>2..5</td><td>Optional 32 bit immediate data</td></tr>
<tr><td>2..9</td><td>Optional 64 bit immediate data</td></tr>
</table>

Data Movement

MOVI/MOVIn

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>6..7</td><td>0 = Reserved, 1=16bits, 2=32 bits 3=64bits</td></tr>
<tr><td>0..5</td><td>Opcode 55=MOVI 56=MOVIn</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved, must be 0</td></tr>
<tr><td>6</td><td>0 = Optional immediate absent,
1= Optional immediate present</td></tr>
<tr><td>4..5</td><td>0=8bit move, 1=16bit mov, 2=32bit move
3=64bit move</td></tr>
<tr><td>3</td><td>0=Operand 1 direct, 1= Operand 1 indirect</td></tr>
<tr><td>0..2</td><td>Operand 1</td></tr>
</table>
</td></tr>
<tr><td>2..3</td><td>16 bit immediate index (Optional)</td></tr>
<tr><td>4..5</td><td>16 bit immediate data</td></tr>
<tr><td>4..7</td><td>32 bit immediate data</td></tr>
<tr><td>4..11</td><td>64 bit immediate data</td></tr>
</table>

Data Movement

PUSH/POP

Opcode	Description
43	PUSH [@] R1 [Imm16]
44	POP [@] R1 [Imm16]
53	PUSHn [@] R1 [Imm16]
54	POPn [@] R1 [Imm16]

Data Movement

Byte	Description
0	<table><tr><th>Bit</th><th>Description</th></tr><tr><td>7</td><td>0= No immediate data 1= Immediate data present</td></tr><tr><td>6</td><td>0=32 bit operand width 1=64 bit operand width</td></tr><tr><td>0..5</td><td>Push or Pop Opcodes</td></tr></table>

Special Register Move

LOADSP SP1,R2

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>6..7</td><td>Reserved, must be 0</td></tr>
<tr><td>0..5</td><td>Opcode =41</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved, must be 0</td></tr>
<tr><td>4..6</td><td>Operand-2. General purpose register</td></tr>
<tr><td>3</td><td>Reserved, must be 0</td></tr>
<tr><td>0..2</td><td>Operand1. Special purpose register</td></tr>
</table>
</td></tr>
</table>

Special Register Move

STORESP R1,SP1

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>6..7</td><td>Reserved, must be 0</td></tr>
<tr><td>0..5</td><td>Opcode =42</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved, must be 0</td></tr>
<tr><td>4..6</td><td>Operand1. Special purpose register</td></tr>
<tr><td>3</td><td>Reserved, must be 0</td></tr>
<tr><td>0..2</td><td>Operand2. General purpose register</td></tr>
</table>
</td></tr>
</table>

Development Environment

- Windows* 32
  - Intel® C Compiler for EFI Byte Code, Version 1.2 Build 20040123
  - Use "C:\Program Files\Intel\EBC\bin\iecvars.bat" to launch EBC compilation and linking environment
- Linker
  - Microsoft* linker Version 7.10.3077 and above

Calling convention

- CDECL only
  - R0: Stack Pointer, R7: return value
  - Like IA32.
    - All parameters are passed through stack including 8-byte structure/long long.
    - Return value is passed through stack if larger than 8-byte
- Between EBC and native
  - EBC ←→ EBC
    - Call/Ret
    - May imply EBC →native→EBC
  - EBC → native code
    - CallEx, VM to handle calling convention
  - Native code → EBC
    - Break 5, to create thunk for address taken EBC functions when EBC image loading
    - One level indirect assignment for EBC function pointer due to the existence of thunking entry

Object / Image format

- Object Format
  - COFF
- Executable Format
  - MS PE32 format

- Segments
  - TEXT, DATA, BSS
  - .CRT$xxx
  - _VARBSS_INIT

Language Examples(1)

```
main()
{
  int *xp, x, y;

  y = 8;
  xp = &y;
  x = *xp;
}
```

```
MOVqw     R0, R0(+0,-16)
MOVIdw    @R0(+0,+8), +8
MOVnw     @R0, R0(+0,+8)
MOVnw     R7, @R0
MOVdw     @R0(+0, +12), @R7
MOVqd     R7, R6
MOVqw     R0, R0(+0,+16)
RET
```

Language Examples(2)

```
int PASS;
long f(long x)
{
  return x+sizeof(int);
}
main()
{
  long x, y;

  x=4;
  y = f(x);
  if ((y+f(x))==16) PASS=1;
  else PASS=0;
}
```

```
f:
  MOVsnw    R7, @R0(+0,+16)
  MOVIqw    R4, 4
  ADD       R7, R4
  RET
```

```
main:
      MOVqw     R0, R0(+0,-48)
      MOVInw    @R0(0,8), (0,4)
      MOVsnw    @R0, @R0(+0,+8)
      CALL      f
      MOVnw     @R0(+0,+24), R7
      MOVsnw    @R0(+0,+16), @R0(+0,+24)
      MOVsnw    @R0, @R0(+0,+8)
      CALL      f
      MOVnw     @R0(+0,+32), R7
      MOVsnw    R7, @R0(+0,+16)
      MOVnw     R4, @R0(+0,+32)
      ADD       R7, R4
      MOVsnw    R7, R7
      CMPIeq    R7, 16
      JMP8cc    $B2$5
      MOVreld   R7, PASS
      MOVIdw    @R7, +1
      JMP8      $B2$6
$B2$5:
      MOVreld   R7, PASS
      MOVdw     @R7, R6
$B2$6:
      MOVqd     R7, R6
      MOVqw     R0, R0(+0,+48)
      RET
```

Demo