Improve LLVM binary utilities

Abstract

LLVM includes binary utilities equivalent to GNU binutils. Basic functionalities are done but there are missing ones like Mach-O support. This project aims to support those missing functionalities and improve usability for those who crave for an alternative to GNU binutils.

This proposal is twofold:

- Improve Mach-O support
- More human-friendly disassembly

Implementation Plan

Fix small bugs (before the official coding period)

Before working on relatively large tasks, I'd like to write small patches to get used to LLVM development. During the community bonding period I'll fix some issues reported on Bugzilla such as:

- Tidy up llvm-ar's error messages: https://bugs.llvm.org/show_bug.cgi?id=40244
- Add support for ANDROID_REL and ANDROID_RELA section types: https://bugs.llvm.org/show_bug.cgi?id=40775

If these bugs are assigned to another developer I'll ask him or her to hand it over to me or look for another one.

llvm-objcopy/llvm-strip: Improve Mach-O support

Currently Reader and Writer for Mach-O are implemented but it does not support objcopy operations including basic functionalities like -j option and -strip-*. I'll implement following functionalities for Mach-O format:

- Print an error message if a flag is not yet implemented.
- --only-section, --remove-section
- --add-section, --set-section-flags, --rename-section
- --strip-all, --strip-dwo
- --dump-section, raw binary support (`-Ibinary` and `-Obinary`)

llvm-objdump: More Human-friendly Disassembly

(This feature is inspired by a feedback on llvm-dev.)

Currently, the output of `llvm-objdump -d` of the following c++ program

```
// example.cc
#include <cstdio>
int counter = 0;

void increment_counter() {
   counter++;
}

void print_counter() {
   printf("current value: %d\n", counter);
}
```

looks like: `$ llvm-objdump -dC example -x86-asm-syntax=intel`

```
example:      file format ELF64-x86-64

Disassembly of section .text:

0000000000400500 increment_counter():
 400500:       55      push    rbp
 400501:       48 89 e5        mov     rbp, rsp
 400504:       8b 04 25 34 10 60 00    mov     eax, dword ptr [6295604]
 40050b:       83 c0 01        add     eax, 1
 40050e:       89 04 25 34 10 60 00    mov     dword ptr [6295604], eax
 400515:       5d      pop     rbp
 400516:       c3      ret
 400517:       66 0f 1f 84 00 00 00 00 00      nop     word ptr [rax + rax]

0000000000400520 print_counter():
 400520:       55      push    rbp
 400521:       48 89 e5        mov     rbp, rsp
 400524:       48 83 ec 10     sub     rsp, 16
 400528:       48 bf f4 05 40 00 00 00 00 00   movabs  rdi, 4195828
 400532:       8b 34 25 34 10 60 00    mov     esi, dword ptr [6295604]
 400539:       b0 00   mov     al, 0
 40053b:       e8 c0 fe ff ff  call    -320 <printf@plt>
 400540:       89 45 fc        mov     dword ptr [rbp - 4], eax
 400543:       48 83 c4 10     add     rsp, 16
 400547:       5d      pop     rbp
 400548:       c3      ret
 400549:       0f 1f 80 00 00 00 00    nop     dword ptr [rax]
```

This somewhat lacks readability. I'd like to work on the more human-friendly disassembly which looks like: `$ llvm-objdump -dC example -x86-asm-syntax=intel -prettify`

```
example:      file format ELF64-x86-64

Disassembly of section .text:

0000000000400500 increment_counter():
 400500:       55                      push    rbp
 400501:       48 89 e5                mov     rbp, rsp
 400504:       8b 04 25 34 10 60 00    mov     eax, dword ptr [0x601034 /* counter */]
 40050b:       83 c0 01                add     eax, 1
 40050e:       89 04 25 34 10 60 00    mov     dword ptr [0x601034 /* counter */], eax
 400515:       5d                      pop     rbp
 400516:       c3                      ret
 400517:       66 0f 1f 84 00 00 00    nop     word ptr [rax + rax]
               00 00

0000000000400520 print_counter():
 400520:       55                      push    rbp
 400521:       48 89 e5                mov     rbp, rsp
 400524:       48 83 ec 10             sub     rsp, 16
 400528:       48 bf f4 05 40 00 00    movabs  rdi, 0x4005f4 /* "current value: %d\n" */
               00 00 00
 400532:       8b 34 25 34 10 60 00    mov     esi, dword ptr [0x601034 /* counter */]
 400539:       b0 00                   mov     al, 0
 40053b:       e8 c0 fe ff ff          call    -320 <printf@plt>
 400540:       89 45 fc                mov     dword ptr [rbp - 4], eax
 400543:       48 83 c4 10             add     rsp, 16
 400547:       5d                      pop     rbp
 400548:       c3                      ret
 400549:       0f 1f 80 00 00 00 00    nop     dword ptr [rax]
```

Features:

- **Syntax highlighting:** Utilize Marked Up Disassembly Output.
  - The current implementation (`MCInstPrinter`) of the feature simply writes marked up disassembly into a string, e.g., "`addl <imm:$1>, <reg:%eax>`".
  - Parsing the output string in llvm-objdump is awkward. Instead, I'll update `MCInstPrinter` so that it prints to a vector of fragments (`MarkedUpSpan`).
  - I'll support x86 in AT&T syntax (`X86ATTInstPrinter`) at first.

```
struct MarkedUpOpcode { std::string OpName; };
struct MarkedUpReg { std::string RegName; };
struct MarkedUpImm { int64_t Value; };
struct MarkedUpMem { std::variant<X86MarkedUpMem, /* ... */> Mem; };
struct X86MarkedUpMem {
  MarkedUpReg X86Base;
  MarkedUpReg X86Index;
  MarkedUpImm X86DispVal;
  MarkedUpImm X86ScaleVal;
};

struct MarkedUpSpan {
  std::variant<MarkedUpOpcode, MarkedUpReg, MarkedUpImm, MarkedUpMem,
               std::string /* others */> Span;
};

class MCInstPrinter {
  /// Print the specified MCInst to the specified raw_ostream.
  virtual void printInst(const MCInst *MI, raw_ostream &OS, StringRef Annot,
                         const MCSubtargetInfo &STI) = 0;
  /// (New!) Print the instruction to the specified MarkedUpSpan vector.
  virtual void printInst(const MCInst *MI, vector<MarkedUpSpan> &O, StringRef Annot,
                         const MCSubtargetInfo &STI) = 0;
};
```

- **Print symbol names:** Search the symbol table for memory addresses.
- **Print ASCII strings in .rodata:** If an immediate value is valid memory address which points to an ASCII string in .rodata, print the string. It is useful when we use a string literal such as the format string of `printf`.
- **Wrap hex dumps of instructions**
- **Emphasise relocations:** The disassembly of a relocation points is confusing. It should emphasise where relocations occur:

```
8b 04 25 ?? ?? ?? ??   mov     eax, dword ptr [0]
         ^ Relocation: counter (0000000000000007: R_X86_64_32S)
```

Timeline

- **Community Bonding Period (May 6 - May 26)**
  - Fix small bugs to get used to the LLVM development.
- **Week 1 (May 27 - June 2)**
  - Improve Mach-O support: Implement --only-section and --remove-section
- **Week 2 (June 3 - June 9)**
  - Improve Mach-O support: Implement --add-section, --set-section-flags, and --rename-section
- **Week 3 (June 10 - June 16)**
  - Improve Mach-O support: support strip options like --stirp-all
- **Week 4 (June 17 - June 23)**
  - Improve Mach-O support: support --dump-section

- *June 24 - June 28: Phase 1 Evaluation*

- **Week 5 (June 24 - June 30)**
  - Improve Mach-O support: Support raw binary (that is, `-Ibinary` and `-Obinary`)
- **Week 6 (July 1 - July 7)**
  - More Human-friendly Disassembly: Fix output format (wrapping hex dump of instructions and indentations)
- **Week 7 (July 8 - July 14)**
  - More Human-friendly Disassembly: Print symbol names
  - More Human-friendly Disassembly: ASCII strings in .rodata
- **Week 8 (July 15 - July 21) & Week 9 (July 22 - July 28)**
  - More Human-friendly Disassembly: Emphasis relocations

- *July 26 - July 28: Phase 2 Evaluation*

- **Week 10 (July 29 - Aug 4)**
  - More Human-friendly Disassembly: Implement `X86ATTInstPrinter::printInst` which outputs a `vector<MarkedUpSpan>`
- **Week 11 (Aug 5 - Aug 11)**
  - More Human-friendly Disassembly: Implement syntax highlighting
    - Implement a printer function in llvm-objdump which takes `vector<MarkedUpSpan>` as input.
- **Week 12 (Aug 12 - Aug 18) & Week 13 (Aug 19 - Aug 26)**
  - The last two weeks are for unpredictable delays. If I finished all tasks as scheduled, I'll work on fixing bugs reported on Bugzilla.
- *Aug 26 - Sept 2: Final Evaluation*