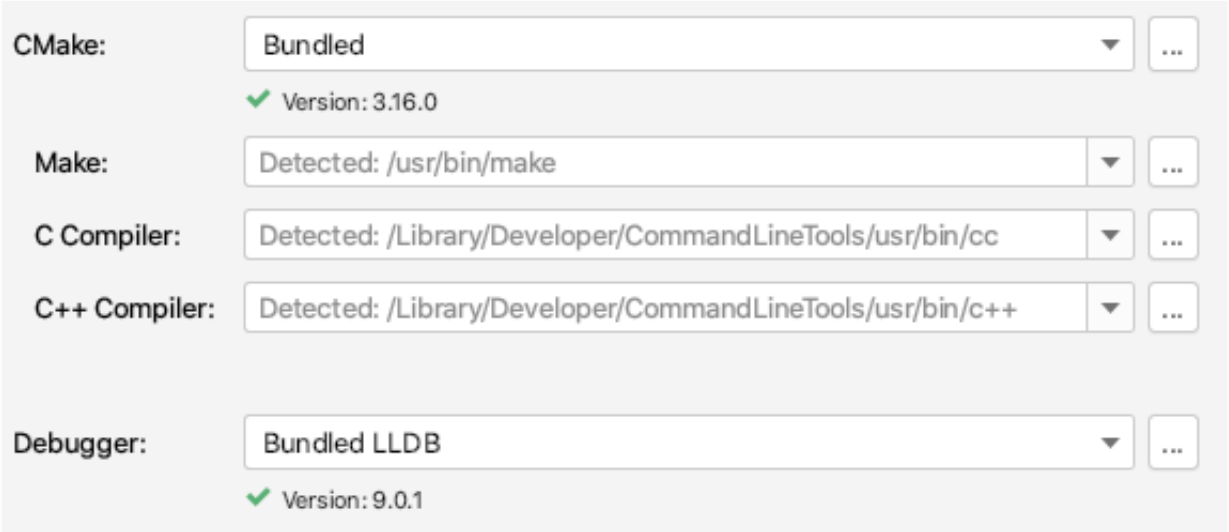
一、获取和编译LLVM-UPT

我的运行环境（cmake过低应该不行）



C和C++编译器版本，本机是mac自带clang，windows或linux可以用gcc

```
wangyilong@wyls-MBP ~/f/l/t/c/cmake> cc -v
Apple clang version 11.0.0 (clang-1100.0.33.8)
Target: x86_64-apple-darwin19.0.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
wangyilong@wyls-MBP ~/f/l/t/c/cmake> c++ -v
Apple clang version 11.0.0 (clang-1100.0.33.8)
Target: x86_64-apple-darwin19.0.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

获取代码

```
git clone https://github.com/Azurewyl/llvm_upt.git
cd llvm && mkdir build && cd build
```

用Ninja构建（如果配置了Ninja，建议，一般LLVM开发人员都会用ninja，否则编译时长太漫长了，不适合开发）

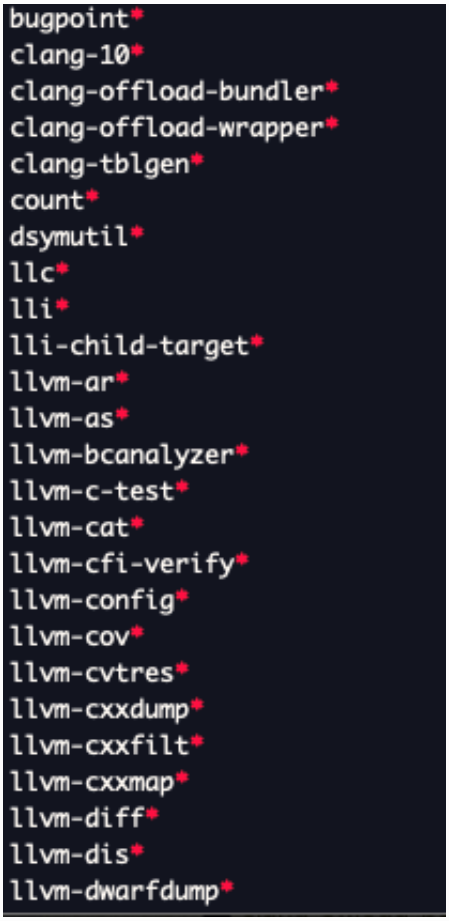
```
cmake -G Ninja -DCMAKE_BUILD_TYPE=Debug -DLLVM_TARGETS_TO_BUILD=UPT ../
ninja
```

用Cmake构建

```
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug -
DLLVM_TARGETS_TO_BUILD=UPT ../
make -j9
```

查看编译出来的执行程序，应当包括clang/llc等：

```
cd build/bin
ls
```



查看后端支持哪些目标：

```
wangyilong@wyls-MBP ~/f/l/c/bin> ./llc -version
LLVM (http://llvm.org/):
  LLVM version 10.0.0svn
  DEBUG build with assertions.
  Default target: x86_64-apple-darwin19.0.0
  Host CPU: skylake

  Registered Targets:
    upt    - UPT, BUPT 32bit Cpu
    x86    - 32-bit X86: Pentium-Pro and above
    x86-64 - 64-bit X86: EM64T and AMD64
```

二、使用和测试方法

LLC

示例LLVM IR代码：

```
define i32 @addi_big(i32 %a) nounwind {
  %1 = add i32 %a, 65536
  ret i32 %1
}
```

LLVM IR 到汇编码

```
./llc -march upt -filetype=asm example.ll -o example.s
```

运行结果：

```
wangyilong@wyls-MBP ~/f/l/c/bin> ./llc -march upt -filetype=asm example.ll -o example.s
wangyilong@wyls-MBP ~/f/l/c/bin> cat example.s
        .text
        .file   "example.ll"
        .globl  addi_big                # -- Begin function addi_big
        .type   addi_big,@function
addi_big:                               # @addi_big
# %bb.0:
        MOVL V0, 0
        MOVU V0, 1
        ADDR V0, A0, V0
        RET
.Lfunc_end0:
        .size   addi_big, .Lfunc_end0-addi_big
                                        # -- End function

        .section        ".note.GNU-stack","",@progbits
```

可以添加--verify-machineinstrs选项，用于启动机器码验证器对指令的操作数数量、操作数寄存器类别和寄存器生存期等相关检查。添加 -show-mc-encoding查看对应的二进制码

```
./llc -march upt -filetype=asm -verify-machineinstrs -show-mc-encoding
example.ll -o example.s
```

运行结果：

```
wangyilong@wyls-MBP ~/f/l/c/bin> ./llc -march upt -verify-machineinstrs -show-mc-encoding example.ll -o example.s
wangyilong@wyls-MBP ~/f/l/c/bin> cat example.s
        .text
        .file   "example.ll"
        .globl  addi_big                # -- Begin function addi_big
        .type   addi_big,@function
addi_big:                               # @addi_big
# %bb.0:
        MOVL V0, 0                      # encoding: [0x00,0x00,0x21,0x40]
        MOVU V0, 1                      # encoding: [0x01,0x00,0x21,0x44]
        ADDR V0, A0, V0                 # encoding: [0x00,0x08,0x22,0x18]
        RET                             # encoding: [0x00,0x00,0xe0,0x4d]
.Lfunc_end0:
        .size   addi_big, .Lfunc_end0-addi_big
                                        # -- End function

        .section        ".note.GNU-stack","",@progbits
```

LLVM IR到可执行文件

```
./llc -march upt -filetype=obj test.ll
```

运行结果并使用hexdump查看内容：

```
wangyilong@wyls-MBP ~/f/l/c/bin> ./llc -march upt -filetype=obj example.ll -o example.o
wangyilong@wyls-MBP ~/f/l/c/bin> hexdump example.o
0000000 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
0000010 01 00 28 00 01 00 00 00 00 00 00 00 00 00 00 00
0000020 b0 00 00 00 00 00 00 00 34 00 00 00 00 00 28 00
0000030 05 00 01 00 00 00 21 40 01 00 21 44 00 08 22 18
0000040 00 00 e0 4d 00 00 00 00 00 00 00 00 00 00 00 00
0000050 00 00 00 00 07 00 00 00 00 00 00 00 00 00 00 00
0000060 04 00 f1 ff 22 00 00 00 00 00 00 00 10 00 00 00
0000070 12 00 02 00 00 2e 74 65 78 74 00 65 78 61 6d 70
0000080 6c 65 2e 6c 6c 00 2e 6e 6f 74 65 2e 47 4e 55 2d
0000090 73 74 61 63 6b 00 61 64 64 69 5f 62 69 67 00 2e
00000a0 73 74 72 74 61 62 00 2e 73 79 6d 74 61 62 00 00
00000b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

CLANG

测试代码

main.c

```c
#include "math.h"
int array[3] = {3, 2, 1};     // 全局数组
int main(void) {
  int res;
  bubbleSort(array, 3); // array = [1, 2, 3]
  res = fib(array[2]);  // res = fib(3) = 2
  res = logic_and_shift(res, 1); // res = 7
  return res;
}
```

math.h

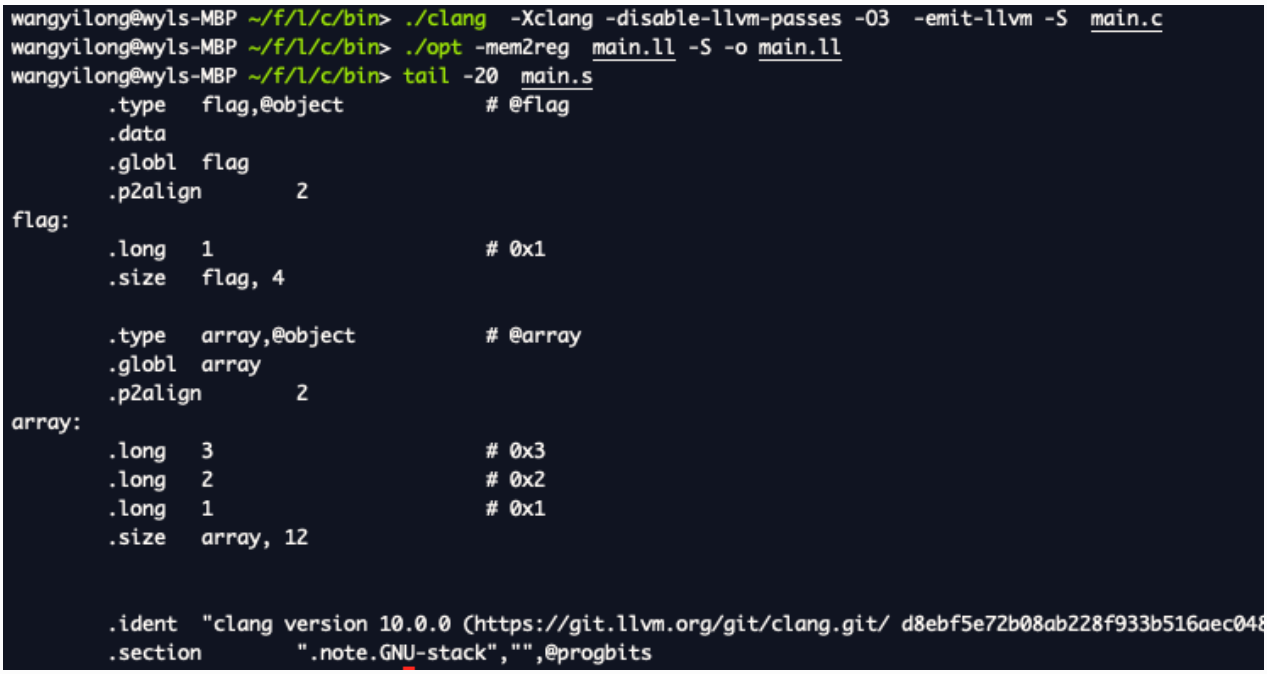
```c
// 冒泡排序
void bubbleSort(int arr[], int n) {
  int i, j, t;
  for (i = 0; i < n - 1; i++)
    for (j = 0; j < n - i - 1; j++)
      if (arr[j] > arr[j + 1]) {
        t = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = t;
      }
}
// 斐波那契数列
int fib(int x) {
  if (x == 0) return 0;
  if (x == 1) return 1;
  return fib(x - 1) + fib(x - 2);
}

// 算术逻辑运算测试
int logic_and_shift(int a, int b) {
  unsigned int c = (a & b);
  unsigned int d = (a | b);
  unsigned int e = (a ^ b);
  return c + (d >> 1) + (e << 1);
}
```

C语言到LLVM IR

无优生成的代码非常的冗余，因此结合LLVM的生态，使用clang和opt编译源程序到LLVM IR,并开启O1级别的优化

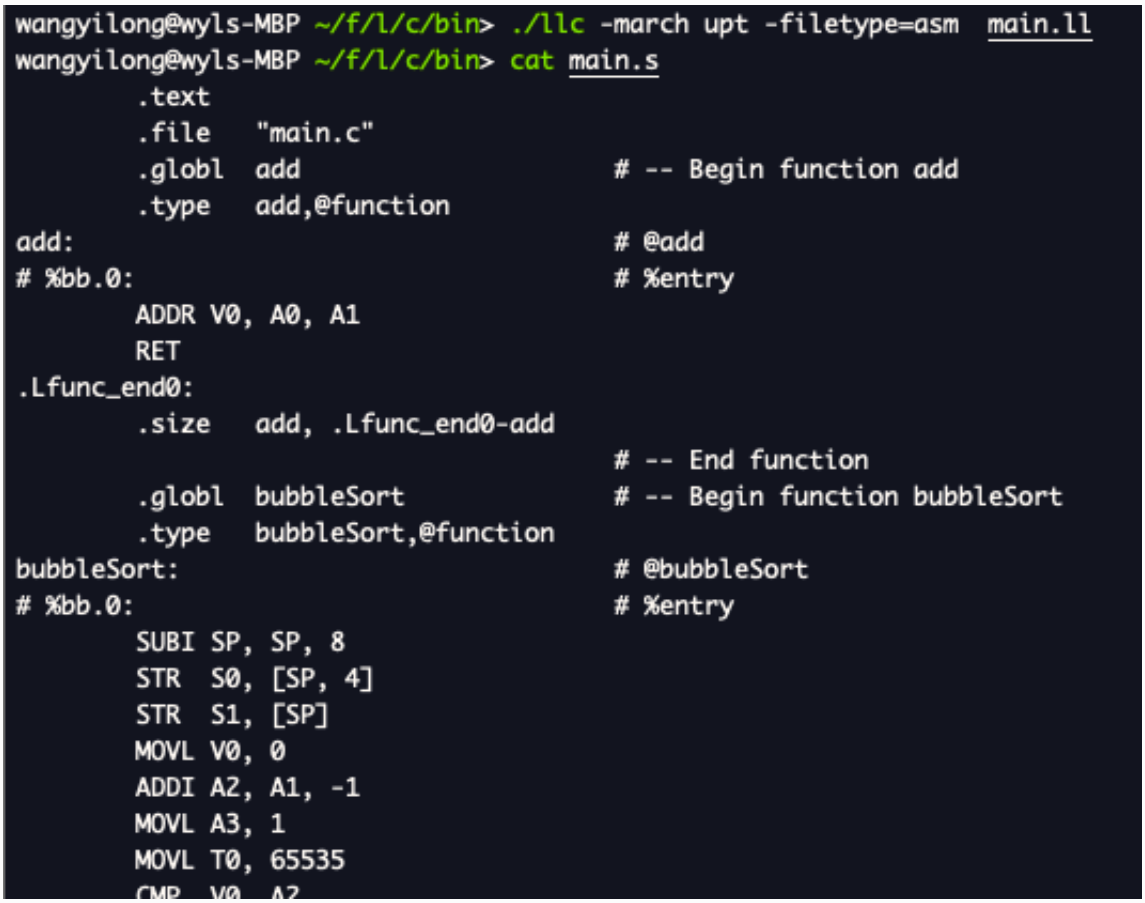
```
./clang -O1 -Xclang -disable-llvm-passes  -emit-llvm -S  main.c
./opt -mem2reg  main.ll -S -o main.ll
```

运行示例：



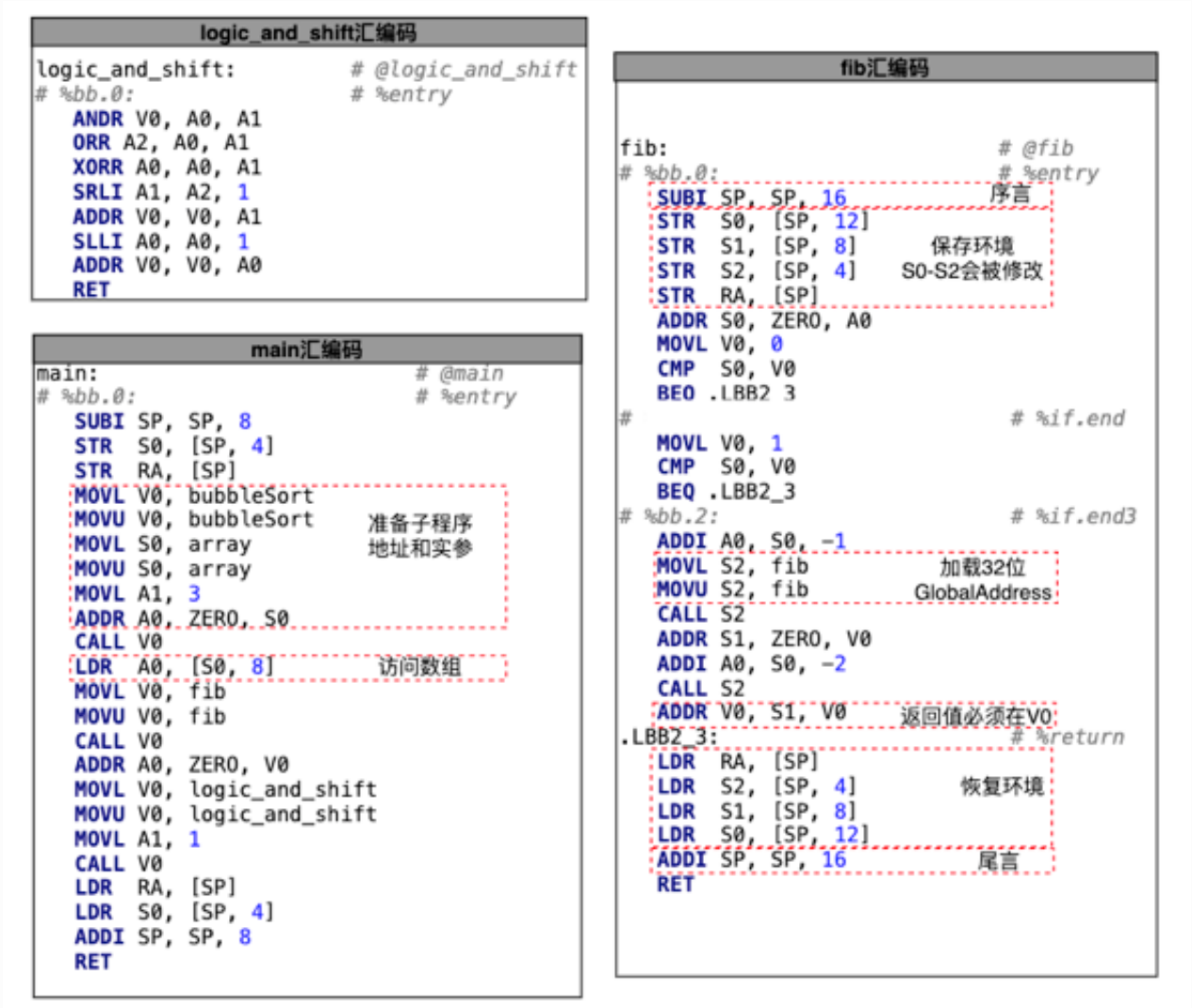
IR到汇编码

再使用llc 编译LLVM IR到汇编码,运行结果：



```
wangyilong@wyls-MBP ~/f/l/c/bin> ./llc -march upt -filetype=asm  main.ll
wangyilong@wyls-MBP ~/f/l/c/bin> cat main.s
        .text
        .file   "main.c"
        .globl  add                     # -- Begin function add
        .type   add,@function
add:                                    # @add
# %bb.0:                                # %entry
        ADDR V0, A0, A1
        RET
.Lfunc_end0:
        .size   add, .Lfunc_end0-add
                                        # -- End function
        .globl  bubbleSort              # -- Begin function bubbleSort
        .type   bubbleSort,@function
bubbleSort:                             # @bubbleSort
# %bb.0:                                # %entry
        SUBI SP, SP, 8
        STR  S0, [SP, 4]
        STR  S1, [SP]
        MOVL V0, 0
        ADDI A2, A1, -1
        MOVL A3, 1
        MOVL T0, 65535
        CMP  V0, A2
```

查看logic_and_shift、fib和main函数的编译结果。限于篇幅，不展示bubbleSort的汇编码：



C语言到汇编码或二进制编码

使用方法

```
./clang -cc1 -triple upt -S test.c  # C =》 汇编码
./clang -cc1 -triple upt test.c     # C=》 二进制可执行文件
```

示例代码：

```
wangyilong@wyls-MBP ~/f/l/c/bin> cat upt_test_set.c
int add(int a,int b){
  return  a + b;
}
int fun(int a,int b){
  a = (a<<2);
  b = (b>>2);
  int c= add(a,b);
  return c;
}

wangyilong@wyls-MBP ~/f/l/c/bin> ./clang -cc1 -triple upt -S  -O1  upt_test_set.c
wangyilong@wyls-MBP ~/f/l/c/bin> ./clang -cc1 -triple upt -O1  upt_test_set.c
wangyilong@wyls-MBP ~/f/l/c/bin> cat upt_test_set.s
        .text
        .file   "upt_test_set.c"
        .globl  add
        .type   add,@function
add:
        ADDR V0, A1, A0
        RET
.Lfunc_end0:
        .size   add, .Lfunc_end0-add
```

LLVM-LIT测试工具

用法：

```
./llvm-lit -v path/to/test/CodeGen/UPT
```

运行示例，测试加法函数addi_big的汇编码和二进制编码的正确性：

```
wangyilong@wyls-MBP ~/f/l/c/bin> cat ../../test/CodeGen/UPT_TEST/alu.ll
; NOTE: Assertions have been autogenerated by utils/update_llc_test_checks.py
; RUN: llc -march upt -verify-machineinstrs -show-mc-encoding <  %s \
; RUN:    | FileCheck %s -check-prefix=ALU

define i32 @addi_big(i32 %a) nounwind {
; ALU-LABEL: addi_big:
; ALU:  # %bb.0:
; ALU-NEXT:     MOVL V0, 0                      # encoding: [0x00,0x00,0x21,0x40]
; ALU-NEXT:     MOVU V0, 1                      # encoding: [0x01,0x00,0x21,0x44]
; ALU-NEXT:     ADDR V0, A0, V0                 # encoding: [0x00,0x08,0x22,0x18]
; ALU-NEXT:     RET                             # encoding: [0x00,0x00,0xe0,0x4d]
  %1 = add i32 %a, 65536
  ret i32 %1
}
```

测试结果通过：

```
wangyilong@wyls-MBP ~/f/l/c/bin> ./llvm-lit -v ../../test/CodeGen/UPT_TEST
-- Testing: 1 tests, 1 workers --
PASS: LLVM :: CodeGen/UPT_TEST/alu.ll (1 of 1)
Testing Time: 0.06s
  Expected Passes    : 1
```

总结

交叉编译和测试

工具	功能	命令或选项
llc	IR => asm	$ ./llc -march upt -filetype=asm test.ll
	IR => obj	$ ./llc -march upt -filetype=obj test.ll
	IR => asm+encoding	$ ./llc -march upt -show-mc-encoding test.ll
	Enable Feature	$ ./llc -march upt -mcpu upt-generic -mattr=+m test.ll
clang	C => IR	$ ./clang -cc1 -triple upt -emit-llvm test.c
	C => asm	$ ./clang -cc1 -triple upt -S test.c
	C => obj	$ ./clang -cc1 -triple upt test.c
llvm-lit	Test	$ ./llvm-lit -v path/to/test/UPT/CodeGen