BITCOIN MINING USERS' GUIDE

Written by Merlin Skinner-Oakes, 1/11/2022

Last updated .

1 DESCRIPTION

This programme implements SHA-256 cryptographic hash functions as used by Bitcoin. It attempts to "mine" a block. This is in no way a useful way to mine Bitcoin, but should be considered more of a working model in the manner of a table-top model steam engine. It was inspired by Ken Shirriff's posts such as http://www.righto.com/2014/09/mining-bitcoin-with-pencil-and-paper.html.

BITCOIN.COM is intended to run under CP/M 2.2. It should run under any such system, though improved progress reporting is only available when run on a ZARC system.

In this document, the following style conventions are used:

```
Machine output
User commands
Comments
```

2 USAGE

To start mining, simply enter the following command from the CP/M prompt:

```
A> bitcoin
```

A number of internal validation tests are performed. This consists of applying the hash function on a number of test messages. The results are compared with known results stored in the code. Any discrepancies are reported.

If the file BITCOIN.SAV is present on the currently logged drive, this is loaded. It contains the search state from a previous session:

- iter_count: number if hashes computed (4 bytes)
- bit_hdr_buf: Bitcoin block header buffer (80 bytes)
- best_hash: smallest hash found (32 bytes)
- best_nonce: nonce corresponding to above (4 bytes)

BITCOIN.SAV may be modified by an external programme if required. If the file cannot be loaded, the Bitcoin genesis block is used and the search started with the nonce value set to zero.

Progress is reported from time to time (every ten seconds on ZARC) and whenever a better (smaller) hash is found.

Press control-C (^C) to exit. BITCOIN.SAV is written to the currently logged drive, overwriting any previous version of this file.

3 SAMPLE RUN

```
A> bitcoin
*** Bitcoin Mining V1.0 ***
Null message test: OK
Short message test: OK
Long message test: OK
Long message test 2: OK
Bitcoin genesis block: OK
Bitcoin example block: OK
Loading state from file D:BITCOIN.SAV
OK

Best nonce / hash found:
1393 0031CFC620EA408D9D03749547E5C8266E59FB766BF071EDA1DCC5C4D1ECCE5A
Starting mining. Press ^C to exit.
Iteration: 4162, delta: 55
Best nonce / hash found:
4200 00055472F907066521D4F41ABD2D1B3CBC102C78C60A153C296771793C50921B
Iteration: 8580, delta: 56
Best nonce / hash found:
8603 00007CC6EC08C5D53C32CEAF6E8309C32EB07FC903A94EC77AC2BF6E48F7ADB3
Iteration: 11285, delta: 55
Saving state to file D:BITCOIN.SAV
Existing file deleted
Returning to CP/M.

D>
```

4 LIMITATIONS

Bitcoin difficulty is not implemented. The search continues indefinitely, reporting each time a “better” (smaller) hash is found. There is also no facility for reporting mined blocks, though in practice this is astronomically unlikely to occur and therefore not an issue.

Performance is extremely limited, as one would expect. ZARC is based around a Z80 CPU running at 8 MHz with no wait states. This achieves approximately 5.5 hashes per second. The algorithm could certainly be optimised, though I suspect even 10 hashes per second would be challenging.

5 FURTHER DEVELOPMENT

The routines in sha-256.z80 may be used for other purposes where SHA-256 is required. These routines use little-endian values internally, as is conventional on Z80 systems. SHA-256 assumes big endian variables, so some ugly endian conversion is involved to implement Bitcoin. This must be considered in other applications. It would be possible to modify the code to work internally with big-endian values, though this would yield only a modest performance improvement. It would also make porting this code onto more modern architectures harder, as these are almost exclusively little-endian.

One optimisation would be to remove redundant calculations. The Bitcoin header is too long to be processed as a single SHA-256 chunk, but only the nonce is changed when mining using this programme. This means that the other chunk is needlessly re-calculated every time. Optimising this would speed up mining significantly, though I have not implemented this.